

A FAST REDUCED-SPACE ALGORITHMIC FRAMEWORK FOR SPARSE OPTIMIZATION

by
Tianyi Chen

A dissertation submitted to Johns Hopkins University in conformity with the
requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
August, 2018

© 2018 by Tianyi Chen
All Rights Reserved

Abstract

Optimization is a crucial scientific tool used throughout applied mathematics. In optimization one typically seeks the lowest value of a chosen objective function from among a set of allowable inputs to that function, i.e., to compute a minimizer. Once an optimization problem is formulated for a particular task, an algorithm for locating a minimizer is employed. For many applications, the optimization problem may possess one or more solutions, some of which may not be desirable from the perspective of the application. In such settings, a popular approach is to augment the objective function through the use of regularization, which should be carefully chosen to ensure that solutions of the regularized optimization problem are useful to the application of interest.

Perhaps the most popular type of regularization is ℓ_1 -regularization in the last two decades. Motivation for incorporating ℓ_1 -regularization is its sparsity-inducing and shrinkage properties, which have demonstrated its utility in improving the interpretation and accuracy of model estimation in both theoretical and practical aspects. Many methods have been proposed for solving ℓ_1 -regularization problems. Roughly, there are first-order methods, which have small computational iteration complexity but often are inefficient on realistic applications, and second-order methods, which have large computational iteration complexity but are robust and efficient in terms of the number of iterations typically required.

In this thesis we present a new second-order framework that aims to balance the strengths of first-order and second-order methods. Specifically, our framework uses a limited amount of second-derivative information by making use of a mechanism for predicting those variables that will be zero at a solution. In this manner, the computational iteration complexity can be controlled. Moreover, by using second-derivative information within certain computed subspaces, our framework is highly efficient and robust in terms of the overall number of iterations typically required. We present numerical comparisons to other state-of-the-art first and second-order methods that validate our approach and further investigate an implementation of our approach that uses parallel computation.

Primary Reader and Advisor: Daniel P. Robinson

Secondary Reader: Amitabh Basu

Acknowledgments

I would foremost like to thank my advisor Daniel P. Robinson for his fundamental role in the completion of this thesis. He selflessly shares his knowledge, guides me to improve myself, and constantly tries to improve my work. Infinite deep gratitude for his invaluable support and guidance, which are crucial for me to move forward on my graduate student career in Johns Hopkins.

I would also like to thank Frank E. Curtis for his guidance and help in our course of research. This work would not be possible to be carried out in the past few years without his expertise and resource. I am honored to collaborate with him and very grateful for his valuable comments, which encouraged me to make improvements.

I would also like to thank Amitabh Basu for being a reader of my thesis and his participation throughout my graduate student career. I sincerely appreciate his time and effort for my thesis. Moreover, many thanks for the priceless treasures that I learned from him.

I would like to acknowledge James C. Spall for being one of the examiners in my dissertation defense. I really appreciate his guidance, time, and help in the past few years. I also want to acknowledge my faculty advisor in the computer science department, Jason Eisner; the success of completing this thesis is inseparable from his encouragement to broaden my knowledge of regularization.

In my whole PhD career, Johns Hopkins University, especially the Depart-

ment of Applied Mathematics and Statistic and the Department of Computer Science provided me with abundant support in every aspect of my life and created a motivational environment where I was able to learn from the surrounding brilliant people, and acquire various knowledge to improve myself everyday. Thank you! The Computational Optimization Research laboratory at Lehigh University (COR@L) supports sufficient computational resources for my numerous heavy experiments. I sincerely appreciate COR@L, especially Martin Takac and Afshin Oroojlooy Jadid who helped me in troubleshooting.

I also want to thank Microsoft and Amazon for offering me attractive research internships and exciting projects where I refined my various capabilities and then came back to push the progress of my thesis. I also thank my friends who brought me a lot of happiness. In the end, I thank my family and girlfriend in China. Without their understanding and support, I would not be where I am today.

Contents

Abstract	ii
Acknowledgments	iii
1 Introduction	1
1.1 Problem Studied	1
1.2 Regularization	3
1.3 Example Applications	6
1.3.1 Binary classification using logistic regression	7
1.3.2 Conditional log-linear model	8
1.4 State-of-the-Art for Sparse Optimization	9
1.5 Overview: Proposed FaRSA Framework	11
1.6 Contributions of the Thesis	12
1.7 Notation	15
1.8 Outline of the Thesis	15
2 Background Material	17
2.1 Convexity	17
2.2 Subdifferential and Subgradients	19
2.3 Constrained and Unconstrained ℓ_1 -Formulations	21

2.4	Properties of ℓ_1 -Regularization	23
2.4.1	Sparsity-inducing property	23
2.4.2	Shrinkage property	26
2.5	An Orthant-Wise Projected Search	28
3	Related Work on ℓ_1-Regularized Optimization Problems	30
3.1	Proximal Algorithms	30
3.1.1	Proximal gradient method	31
3.1.1.1	Iterative Shrinkage-Thresholding Algorithm . . .	31
3.1.1.2	Fast Iterative Shrinkage-Thresholding Algorithm	32
3.1.2	Proximal Newton method	33
3.2	Orthant Based Methods	34
3.2.1	Orthant-Wise Limited-Memory Quasi-Newton Method . .	35
3.2.2	Orthant-Based Adaptive Method	36
3.3	Equivalent Smooth Reformulations	38
4	FaRSA: A Solver for ℓ_1-Regularized Optimization Problems	40
4.1	Algorithmic Framework	41
4.1.1	Optimality measures	42
4.1.2	Variable-freeing step	49
4.1.3	Objective-reducing step	50
4.1.4	Practical enhancements	52
4.1.5	The complete algorithm	53
4.2	Convergence Analysis	54
4.2.1	Global convergence analysis	56
4.2.2	Local convergence analysis	67
5	Numerical Experiments	77
5.1	FaRSA Software	78

5.1.1	Implementation details	78
5.1.2	Objective functions supported	80
5.1.2.1	Generic objective function	80
5.1.2.2	Logistic loss	81
5.1.2.3	Least-squares loss	85
5.2	Solvers Tested	90
5.3	Experimental Setup	92
5.4	Data Set Description	93
5.5	Experimental Results	104
5.5.1	Runtime and final objective function value	105
5.5.1.1	Matlab versus C implementations of FaRSA . . .	106
5.5.1.2	Comparison of C/C++ solvers	107
5.5.1.3	Comparison of Matlab solvers	110
5.5.2	Local convergence performance of FaRSA	114
5.5.3	Testing accuracy for solutions produced by FaRSA	119
6	Extensions	130
6.1	The ℓ_1/ℓ_p -Regularization Problem	131
6.2	Proximal Operator for the ℓ_1/ℓ_p -Norm	137
6.3	FaRSA for ℓ_1/ℓ_p -Regularization Problem	140
6.3.1	Optimality measure	140
6.3.2	Complete algorithm	141
7	FaRSA: Shared-Memory Multi-core Parallelization	143
7.1	Background of Parallel Programming	144
7.1.1	Process interaction	144
7.1.2	Task decomposition	145
7.2	Prior Parallel Algorithms for ℓ_1 -Problems	146

7.3	FaRSA: Multi-Core and Shared-Memory	147
7.4	Numerical Experiments	149
7.4.1	Experimental setup	149
7.4.2	Numerical results for multi-core FaRSA	150
7.4.3	Multi-core FaRSA versus multi-core LIBLINEAR	156
8	Conclusion	163
	Bibliography	165
	Vita	180

List of Tables

5.1	Resource requirements for when H is formed and saved (“Saving H ”) versus when it is used via matrix-vector products (“Not solving H ”).	89
5.2	Set of binary classification datasets considered. They are organized in ascending order based on file size measured in megabytes (MB).	105
5.3	The required computing time, final objective function value, and the number of iterations for our C and Matlab implementations of FaRSA on the small-scale datasets listed in Table 5.2.	107
5.4	The required computing time and final objective function value for our C implementation of FaRSA, LIBLINEAR, and ASACG on the datasets listed in Table 5.2.	109
5.5	The required computing time and final objective function value for our C implementation of FaRSA, FISTA (SPAMS), and OWL-QN on the datasets listed in Table 5.2.	111
5.6	The computing time and final objective values for our Matlab implementation of FaRSA, OBA, and IRPN on the datasets listed in Table 5.2.	113

5.7	The computing time and final objective function value for our Matlab implementation of FaRSA, PSSas, and PSSgb on the datasets in Table 5.2.	114
5.8	The computing time and final objective value for our Matlab implementation of FaRSA, Gauss-Seidel, and Shooting on the datasets in Table 5.2.	115
5.9	The values of $\ \beta(x_k)\ $ and $\ \phi(x_k)\ $ for the last 5 iterations.	116
5.10	Testing datasets for binary classification.	120
7.1	The final objective function value for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-6}$	150
7.2	The number of iterations for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-6}$	150
7.3	The final objective function value for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-4}$	153
7.4	The number of iterations for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-4}$	153
7.5	The final objective function value for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-2}$	153
7.6	The number of iterations for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-2}$	153

List of Figures

2.1	Geometry of ℓ_1 -regularization and ℓ_2 -regularization.	25
5.1	Required Memory (GB) for saving an n -by- n dense matrix of type double: $\text{mem}(GB) = \frac{8n^2}{1024^3}$. One double variable needs 8 bytes in memory.	86
5.2	Accuracy (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).	122
5.3	Accuracy (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).	123
5.4	Precision (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).	124
5.5	Precision (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).	125
5.6	Recall (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).	126
5.7	Recall (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).	127
5.8	F1-score (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).	128
5.9	F1-score (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).	129

7.1	Underlying architectures for shared-memory model.	145
7.2	Speedup of FaRSA for termination tolerance level $\text{tol} = 10^{-6}$	152
7.3	Speedup of FaRSA for termination tolerance level $\text{tol} = 10^{-4}$	154
7.4	Speedup of FaRSA for termination tolerance level $\text{tol} = 10^{-2}$	155
7.5	Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time (seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-6}$ and using different numbers of cores.	157
7.6	Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time (seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-6}$ and using different numbers of cores.	158
7.7	Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time(seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-4}$ and using different numbers of cores.	159
7.8	Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time(seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-4}$ and using different numbers of cores.	160
7.9	Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of on wall time (seconds) and final objective function val- ues, with a termination tolerance of $\text{tol} = 10^{-2}$ and using different numbers of cores.	161

7.10 Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time (seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-2}$ and using different numbers of cores.	162
--	-----

Chapter 1

Introduction

1.1 Problem Studied

Optimization is a crucial scientific tool in applied mathematics [1, 2, 3, 4]. In optimization one typically seeks a minimizer from among a set of allowable inputs to a chosen function such that this function reaches its lowest value [5]. After formulating an optimization problem for a particular task of interest, locating such a minimizer is achieved by employing an algorithm. For many applications, the optimization problem may produce one or more minimizers that are not desirable from the perspective of the application. In such settings, a popular approach is to augment the objective function through the use of regularization [6, 7, 8, 9, 10], which, in short, should be chosen to ensure that solutions of the regularized optimization problem are useful to the application.

The concept of regularization is central in many scientific domains, e.g., mathematics, statistics and computer science, particularly in the fields of machine learning [11, Chapter 3.1.4] and inverse problems [12, Chapter 1]. As mentioned above, regularization generally refers to the modification of the optimization problem of interest so that minimizers will satisfying certain de-

sirable properties for the application. One of the best-known examples is ℓ_1 -regularization [8], i.e., augmenting the objective function by adding the ℓ_1 -norm of the variables to be optimized, which has received special attention in the last two decades. The ℓ_1 -regularization has sparsity-inducing and shrinkage properties [8], which has demonstrated its utility in improving the interpretation and accuracy of model estimation in both theoretical and practical aspects [8, 13, 14].

Regularization, such as ℓ_1 -regularization, can easily be integrated into the optimization problem formulation [15], usually by including a weighting parameter that controls the level of influence the regularizer has on the optimization problem. Not surprisingly, successful application of ℓ_1 -regularization depends on an effective optimization solver. In the past years, many methods for solving ℓ_1 -regularized problems have been developed [16, 17, 18, 19, 20], and all of them have strengths and weaknesses. In this thesis, we build a new optimization framework (named FaRSA) for minimizing an ℓ_1 -regularized convex and twice continuously differentiable function f that, for a given value of the regularization parameter $\lambda > 0$, is of the form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) + \lambda \|x\|_1 \tag{1.1}$$

with the aim of eliminating the weaknesses of state-of-the-art methods; we also provide publicly available software that outperforms the state-of-the-art, design an extended FaRSA to more complicated structured-sparsity problem, and publish accelerated FaRSA software on multi-core shared-memory system.

The remainder of this chapter is organized as follows. First, in Section 1.2, we briefly review the concept of regularization, and compare its different variants. Next in Section 1.3, we move on to present some popular settings and

applications for ℓ_1 -regularization, some of which will be used in the numerical experiments presented in Chapter 5. In Section 1.4, we provide an overview of the existing solvers for ℓ_1 -regularization problems, followed by an overview of our proposed framework FaRSA (a Fast Reduced-Space Algorithm) in Section 1.5. We summarize the main contributions of this thesis in Section 1.6, present the notation used throughout the thesis in Section 1.7, and finally conclude this chapter with an outline of the remainder of the thesis in Section 1.8.

1.2 Regularization

In many fields, building an appropriate model is crucial for describing and predicting inherent or exterior phenomena and patterns [11]. For a given model, there often exist many parameters with unknown true values to be determined. To estimate these unknown parameters, some data fitting techniques [21] compute appropriate values for the parameters by optimizing a fitting function that is defined by using data measurements, e.g., minimizing a prediction error such as a negative likelihood function [22]. Slightly more formally, if we denote the optimization vector to be optimized by x and the optimal (true) parameter vector by x_{true} , and let f be the objective function for which smaller values for $f(x)$ correspond to better choices for x , then determining the optimal parameters may be formulated as the optimization problem

$$\underset{x \in \mathcal{X}}{\text{minimize}} \quad f(x) \tag{1.2}$$

where \mathcal{X} is the set of allowed parameter values. One popular setting is the unconstrained case, which corresponds to $\mathcal{X} = \mathbb{R}^n$. Once an estimator x^o is computed as a solution to (1.2), there remains the question of how well it performs in terms of the application under consideration. In particular, the following

observations are prevalent in the literature.

- The estimator x^o often has low bias and large variance [23], which often means that a relatively high error is observed when the learned parameters x^o are used for prediction on unseen data. In machine learning, low bias and high variance estimators may cause overfitting and thus be bad estimators.
- An important concept in many fields is that of parsimony. In short, parsimony is essentially the idea that users usually prefer models that are easily interpretable and/or cheaper to use once parameter values are obtained. Therefore, parsimony is often related to obtaining sparse approximate minimizers of f , i.e., minimizers that have relatively few nonzero entries. Unfortunately, the estimator x^o computed as a solution to (1.2) tends to be fully dense, i.e., all entries of x^o are non-zero, which is not very parsimonious as described above.

A common strategy for addressing the first issue above is ℓ_2 -regularization [6, 7, 24, 25]. In particular, one strategy for using ℓ_2 -regularization is—instead of minimizing the measurement function $f(x)$ —to optimize the sum of $f(x)$ and the squared two-norm of x , i.e., choose a positive weighting scalar λ and solve

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) + \lambda \|x\|_2^2. \quad (1.3)$$

For simplicity, let us assume that a solution to the optimization problem (1.3) exists, call it x^{ℓ_2} , and that it is unique.

The ℓ_2 -regularization approach has a Bayesian interpretation, namely as adding the prior that the elements of x are normally distributed. The effect of this prior has a shrinkage property related to λ so that x^{ℓ_2} tends to have a smaller magnitude when compared to x^o . This shrinkage property has been demonstrated to sacrifice bias a bit (make it larger) but also to reduce variance [26]. A larger value for λ causes greater shrinkage and, as a consequence,

often a greater reduction in the variance. Thus ℓ_2 -regularization usually improves the suitability of the model obtained from the learned parameter vector x^{ℓ_2} . However, ℓ_2 -regularization does not usually improve model interpretability because although x^{ℓ_2} converges to zero as $\lambda \rightarrow \infty$, it is rarely the case that any single entry in x^{ℓ_2} is exactly zero for any finite value of λ .

Although *subset selection* is one way to improve model interpretability, it is a discrete process that is often sensitive to changes in the data set [27]; small changes in the data can result in significantly different models. Also, the problem of choosing an optimal subset of variables based on many optimization criteria is NP-hard [28], which can be a problem, practically speaking.

Since both ℓ_2 -regularization and subset selection have drawbacks as just described, an alternative based on ℓ_1 -regularization has been proposed that maintains their strengths and avoids their disadvantages. The first use of ℓ_1 -regularization, the *Lasso* [8], was formulated as a constrained problem:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad \sum_{i=1}^n |x_i| \leq t \quad (1.4)$$

for some $t > 0$ with $f(x)$ as a convex function; note that the constraint is precisely $\|x\|_1 \leq t$. This ℓ_1 -penalty constrained problem can be transformed, and then solved, as an unconstrained problem, which is a well-known ℓ_1 -regularization problem; we will discuss the transformation in more detail in Section 2.3. Importantly, ℓ_1 -regularization has sparsity-inducing and shrinkage properties [8]. The sparsity-inducing property has the effect (as discussed earlier) of improving the interpretability of the recovered model and allowing accurate predictions on new data by sacrificing bias to reduce variance. Thus, ℓ_1 -regularization simultaneously achieves subset selection (those parameters that are nonzero) and better model generalization via a bias-variance tradeoff.

Let x^{ℓ_1} be the solution for (1.4). It follows that x^{ℓ_1} lies at the intersection of the feasible set $\{x : \|x\|_1 \leq t\}$ and the lowest level set of f that has nonempty intersection with this feasible set. Geometrically, the feasible set is a polytope because of the manner in which the ℓ_1 -norm is defined, and has vertices, edges, or faces at which one or more entries in x are zero. Informally, since the lowest contour of f that intersects the feasible region is likely to intersect at such locations, ℓ_1 -regularization promotes sparsity. The positive scalar t controls the number of nonzero entries in x^{ℓ_1} . In general, smaller values of t produce solutions x^{ℓ_1} with smaller magnitude and high sparsity, i.e., few nonzero entries.

1.3 Example Applications

Problems of the form (1.1) arise in statistics, signal processing, and machine learning, and are often associated with data fitting or maximum likelihood estimation. A popular setting is binary classification using logistic regression (where f is a logistic cost function), although instances of such problems also arise when performing multi-class logistic regression, for example. Another popular setting is general log-linear models. Instances of (1.1) also surface when using Lasso or elastic-net formulations to perform data analysis and discovery, e.g., in the clustering of data drawn from a union of subspaces [29, 30, 31, 32].

In the next two subsections, we present two popular applications of ℓ_1 -regularization in classification. Specifically, we first discuss ℓ_1 -regularization for binary classification using logistic regression (where f is a logistic cost function), and then describe conditional log-linear models.

1.3.1 Binary classification using logistic regression

In regression, given N data pairs $\{(d_i, y_i)\}_{i=1}^N$, where $d_i = (d_{i,1}, \dots, d_{i,n})^T \in \mathbb{R}^n$ are regressors for the i -th observation and $y_i \in \mathbb{R}$ is the response for the i -th observation, the goal is to build a model that describes the relationship between the response vector $y = (y_1, y_2, \dots, y_N)^T \in \mathbb{R}^N$ and the design matrix $D = [d_1, d_2, \dots, d_N]^T \in \mathbb{R}^{N \times n}$. After the model is learned, it is used to predict the response to new regressors. There exists various models to describe such relationships. The most well-known method is ordinary least squares (OLS) or linear least squares estimation, where we minimize the sum of the squares of the residuals produced by employing a linear model. In the case that ℓ_2 -regularization is used to enhance OLS, the approach is often called Tikhonov regularization or ridge regression [7, 24, 25]. As expected, ℓ_1 -regularization is also commonly used, in which case it is often called a Lasso [8] problem.

For binary classification, each y_i can only take values in the discrete set $\{-1, +1\}$, and one of the most popular binary classification methods is logistic regression [11, Chapter 4.3.2]. Logistic regression is a method that measures the relationship between the categorical dependent variable (e.g., $y_i \in \{-1, +1\}$) and one or more independent variables (e.g., $d_i = (d_{i,1}, \dots, d_{i,n})^T$), by estimating probabilities using a logistic function, which is based on

$$p(y_i \mid d_i, x, b) = \frac{1}{1 + \exp(-y_i(x^T d_i + b))}. \quad (1.5)$$

The training procedure for logistic regression is maximum likelihood estimation. Specifically, we minimize the average negative log-likelihood, which is

$$f(x, b) = \frac{1}{N} \sum_{i=1}^N \log(1 + e^{-y_i(x^T d_i + b)}). \quad (1.6)$$

Unlike the least-squares objective function, in general there is no closed-form solution for the parameter vector that minimizes the logistic regression objective function in (1.6). Thus a robust and efficient solver is needed to carry out an approximate or exact estimation of minimizers of the logistic regression objective function. Once a solution is obtained, predicting the category of a new data point is performed by comparing the probabilities associated with each class via (1.5), and assigning the label with the largest probability.

The ℓ_1 -regularization problem for logistic regression can then be written as:

$$\underset{x \in \mathbb{R}^n, b \in \mathbb{R}}{\text{minimize}} \frac{1}{N} \sum_{i=1}^N \log(1 + e^{-y_i(x^T d_i + b)}) + \lambda(\|x\|_1 + \|b\|_1) \quad (1.7)$$

for some chosen value $\lambda > 0$ for the regularization parameter. Based on our earlier discussion, we know that solutions of problem (1.7) tend to be sparse when λ is relatively large, which has the effect of selecting the variables that are “most important” in terms of the regression problem.

1.3.2 Conditional log-linear model

For the same set of data-pairs in Section 1.3.1, maximum entropy modeling is an alternative estimation technique. The principles behind maximum entropy modeling is as follows: (i) Do not assume anything about non-observed events, which is unlike the assumption of cumulative logistic distribution for logistic regression; and (ii) Find the most uniform (maximum entropy) probability distribution that matches the observations. The most popular use of maximum entropy modeling is conditional probability modeling. Interestingly, maximum entropy modeling on conditional probability has a nice closed-form

solution [33, 34], called the conditional log-linear model, which is defined as

$$p(y_i | d_i, x) = \frac{\exp(x^T d_i)}{Z(x)} \quad (1.8)$$

where $Z(x)$ is a normalization to ensure that the probabilities over all possibilities sum to one; for conditional probability distribution $Z(x) = \sum_j \exp(x^T d_j)$.

The procedure of training a maximum entropy model is equivalent to minimizing the negative average log-likelihood with probabilities estimated by the conditional log-linear model, namely

$$f(x) = -\frac{1}{N} \sum_{i=1}^N \log p(y_i | d_i, x) \quad (1.9)$$

where $p(y_i | d_i, x)$ is defined in (1.8). As for logistic regression, ℓ_1 -regularization on the conditional log-linear model produces sparse solutions that retain the features that exhibit the strongest influence on the probability distribution.

1.4 State-of-the-Art for Sparse Optimization

For solving the ℓ_1 -regularization convex problem (1.1), there exist many first-order optimization methods (i.e., those that only use first-order derivatives of f) including ISTA, FISTA, and SpARSA [20, 35]. First-order methods have proved quite useful because of their simplicity and good performance. Nonetheless, first-order methods are somewhat limited in terms of a lower convergence rate and less reliability for solving ill-conditioned problems, which can often be overcome by employing second-order derivative information.

Second-order methods, i.e., those that involve second-order derivatives of f , have also been proposed, which can roughly be split into three classes: continuously differentiable bound-constrained reformulations [36, 37, 38, 39, 40,

41, 42, 43], proximal-Newton methods [44, 45, 46, 47, 18] and orthant-based methods [19, 48, 17]. Continuously differentiable bound-constrained reformulations transform the unconstrained non-differentiable problem (1.1) into a larger but equivalent bound-constrained continuously differentiable optimization problem (in the sense that a solution to the bound-constrained formulation directly emits a solution to (1.1)), which may be solved using standard bound-constrained optimization solvers such as those cited above. The other two classes more directly attack problem (1.1). In particular, proximal-Newton methods solve problem (1.1) by minimizing a sequence of subproblems formed as the sum of a quadratic approximation to f and the non-smooth ℓ_1 -norm regularizer. For example, the state-of-the-art software LIBLINEAR, which implements newGLMNET [18], uses a coordinate descent algorithm to approximately minimize each piecewise quadratic subproblem. Orthant-based methods, on the other hand, minimize smooth quadratic approximations to (1.1) over a sequence of orthants in \mathbb{R}^n until a solution is found. Of particular interest is the recently proposed orthant-based method OBA [17] in which every iteration consists of a corrective cycle of orthant predictions and subspace minimization steps. Finally we remark that both LIBLINEAR and OBA are valuable state-of-the-art algorithms that complement each other.

Second-order methods can often be superior to first-order methods, in theory, and currently are employed in many machine learning packages, e.g., scikit-learn [49] and Amazon Machine Learning [50]. Nonetheless, the use of second-derivatives comes with computational challenges. For the proximal Newton method that LIBLINEAR is built upon, the computational overhead is controlled by using a coordinate descent (CD) algorithm to approximately minimize each piecewise quadratic function that defines each subproblem. The use of CD means that one should expect excellent performance on problems whose

Hessian matrices are strongly diagonally dominant. However, not all applications have associated Hessian matrices with such nice properties, which means that LIBLINEAR often performs poorly on such problems. On the other hand, the method OBA uses an “active-orthant” based strategy to defined quadratic subproblems that are *smaller* in dimension than the ambient space. In doing so, OBA is able to limit the computation needed to approximately minimize each quadratic subproblem.

1.5 Overview: Proposed FaRSA Framework

In this thesis, we present a new Fast Reduced-Space Algorithm (FaRSA) for solving problem (1.1). FaRSA is designed to capitalize on the advantages of LIBLINEAR and OBA while avoiding their disadvantages. Motivated by the discussion in the previous section, from our perspective the ideal algorithm for solving ℓ_1 -regularized problems should possess the following attributes: (i) It should be able to quickly predict the correct zero/nonzero elements in the minimizer (i.e., what orthant face the solution belongs to) (ii) It should use second-order information in a manner that accelerates convergence, but in a controlled manner so that the computational cost does not become overwhelming; and (iii) It should be as flexible as possible so that future advances in other related areas may be integrated, e.g., advances in the conjugate gradient (CG) method and coordinate descent (CD) method as a tool for approximately minimizing strictly convex quadratic functions.

To realize the above targeted features, FaRSA consists of newly designed orthant-wise learning strategies that generate an evolving set of indices corresponding to variables that are predicted to be nonzero at a solution (i.e., the support). The orthant face prediction of FaRSA is efficient and can achieve

active-set identification after sufficiently many iterations. For acceleration purpose, we utilize a reduced-space quadratic subproblem for which approximate solutions can be computed efficiently. Although similar subproblems are used by OBA, the precise manner in which they are formulated as well as the conditions used to terminate their solution are different. To ensure convergence of our framework, we combine a new projected backtracking line search procedure, an approximate subspace minimization scheme, and a mechanism for determining when the support of the solution estimate should be updated. Additionally, our framework is flexible. In particular, we introduce a new set of conditions that signal how accurately each quadratic subproblem should be solved and allow for various subproblem solvers to be used. In so doing, our method easily accommodates a CG subproblem solver as in OBA and a CD solver as in LIBLINEAR. Interestingly, in theory this allows for multiple subproblem solvers to be used in parallel, thus allowing for numerical performance that can be as good as either LIBLINEAR and OBA regardless of whether the problem Hessians are strongly diagonally dominant. We also develop and publish a version of our FaRSA software that is written in C. As demonstrated in the numerical experiments described in this thesis, FaRSA generally outperforms other state-of-arts solvers in both efficiency and reliability on test problems from the LIBSVM repository.

1.6 Contributions of the Thesis

Below, we briefly summarize the main contributions and results of this thesis. We remark that the bulk of this work has been published in [51] and [52].

- **A Fast Reduced-Space Algorithm (FaRSA)**

In Chapter 4.1, we present a new active-set line search method called FaRSA

for solving the ℓ_1 -regularized convex optimization problem (1.1). The main features of the new method include: (i) an evolving set of indices corresponding to variables that are predicted to be nonzero at a solution (i.e., the support); (ii) a reduced-space subproblem defined in terms of the predicted support; (iii) conditions that determine how accurately each subproblem must be solved, which allow for Newton, linear conjugate gradient, and coordinate-descent techniques to be employed; (iv) a computationally practical condition that determines when the predicted support should be updated; and (v) a reduced proximal gradient step that ensures sufficient decrease in the objective function when it is decided that variables should be added to the predicted support.

- **Global convergence of FaRSA**

In Chapter 4.2.1, we show that FaRSA can achieve global convergence to a solution of problem (1.1). In general, FaRSA achieves global convergence by combining a new projected backtracking line search procedure, an approximate subspace minimization scheme, and a mechanism for determining when the support of the solution estimate should be updated. There are two main theoretical results in this part. The first states that FaRSA guarantees termination in a finite number of iterations if a prescribed nonzero tolerance of accuracy for the solution is assigned. The other one shows that FaRSA generates iterates that converge to the solution of problem (1.1) if the target tolerance of accuracy is zero. Numerical results shown in Chapter 5 validate the effectiveness of FaRSA.

- **Superlinear local convergence of FaRSA**

In Chapter 4.2.2, we prove local convergence properties of the iterates computed by FaRSA. We prove that once an iterate is sufficiently close to a solution, the following two local results hold: (i) The iterates generated by FaRSA pro-

vide active-set identification, i.e., the correct orthant-face of the solution will be identified by the computed iterates. (ii) Once active-set identification occurs, the iterates of FaRSA converge to the solution at a superlinear rate.

- **Publicly available single-core software**

We develop an optimized FaRSA software running on single-core system that is freely available to the public. In fact, we have two versions available with one written in C and the other written in Matlab. In Chapter 5, FaRSA is tested on a number of datasets from the LIBSVM repository, and compared with other state-of-art solvers. The numerical results indicate that FaRSA generally outperforms other state-of-the-art software in both efficiency and reliability. Moreover, metrics related to machine learning applications, e.g., accuracy, precision, and recall, are computed and demonstrate the effectiveness of the solutions returned by the FaRSA software.

- **Extended FaRSA on structured-sparsity problem**

In Chapter 6, we present an extension of FaRSA for a class of structured-sparsity problems, i.e., mixed ℓ_1/ℓ_p -regularization problem. Generally speaking, we at first provide some necessary theoretical analysis about mixed ℓ_1/ℓ_p -norm, and then extend each component of FaRSA to the new target problem.

- **Publicly available multi-core software**

In terms of the exploration of high-dimensional datasets, we also develop an optimized parallel FaRSA software running on multi-core share-memory system that is freely available to the public. In Chapter 7, FaRSA on multi-core shared-memory system is tested on a big-data-set collection from the LIBSVM repository and compared with other state-of-the-art solver. The experimental results show that FaRSA can achieve significant acceleration by multi-core parallelization and performs superior than state-of-the-art solver.

1.7 Notation

We let \mathbb{R} denote the set of real numbers. Given a positive integer n , we use \mathbb{R}^n to denote the n -dimensional vector with real entries. Let $\mathcal{I} \subseteq \{1, 2, \dots, n\}$ denote an index set of variables. For any $v \in \mathbb{R}^n$, we let $[v]_{\mathcal{I}}$ denote the sub-vector of v consisting of elements of v with indices in \mathcal{I} . Similarly, for any symmetric matrix $M \in \mathbb{R}^{n \times n}$, we let $[M]_{\mathcal{I}, \mathcal{I}}$ denote the sub-matrix of M consisting of the rows and columns of M that correspond to the index set \mathcal{I} . In general, $F(x)$ as defined in (1.1) is not differentiable because of the term $\|x\|_1$. However, if the index set \mathcal{I} satisfies $[x]_i \neq 0$ for all $i \in \mathcal{I}$, then we let $\nabla_{\mathcal{I}} F(x)$ denote the vector of partial derivatives of F at x with respect to the coordinates in \mathcal{I} , and $\nabla_{\mathcal{I}\mathcal{I}}^2 F(x)$ denote the matrix of second-order partial derivatives of F at x with respect to pairs of coordinates in \mathcal{I} . For any vector v , we let $\text{sgn}(v)$ denote the vector of the same length as v whose i th component is 0 when $[v]_i = 0$, is 1 when $[v]_i > 0$, and is -1 when $[v]_i < 0$. For any vector v , we let $\|v\|_1$ and $\|v\|$ denote its ℓ_1 -norm and ℓ_2 -norm, respectively. Finally, we use the notation $\{v_k\}_{k \in \mathcal{S}} \rightarrow \bar{v}$ and $\lim_{k \in \mathcal{S}} v_k = \bar{v}$ interchangeably to mean the same thing, namely that $\{v_k\}_{k=1}^{\infty}$ is a sequence, \mathcal{S} is an infinite subsequence of the integers, and $\{v_k\}_{k=1}^{\infty}$ converges to \bar{v} along the subsequence \mathcal{S} as k converges to infinity.

1.8 Outline of the Thesis

In Chapter 2, we give general background, while in Chapter 3 we discuss related work for solving (1.1). In Chapter 4, we present our new Fast Reduced Space Algorithm (FaRSA), prove global convergence and a local convergence rate. In Chapter 5, numerical experiments demonstrate the effectiveness of FaRSA on datasets from the LIBSVM repository, and provide comparisons to state-of-art solvers. Next, in Chapter 6 we discuss how the ideas underpin-

ning FaRSA may be extended to problem formulations beyond those using ℓ_1 -regularization. Then in Chapter 7 parallel acceleration of FaRSA on multi-core shared-memory system is discussed. Finally we conclude the thesis with a conclusion Chapter 8.

Chapter 2

Background Material

To help readers understand the main content of the following chapters, we present background knowledge in this chapter. Expert readers may choose to skip this chapter. We organize the remainder of this chapter as follows. In Section 2.1, we briefly review the definitions of convex sets and convex functions. In Section 2.2 we discuss the important concept of the subdifferential, as well as its importance in terms of optimization. Next, in Section 2.3 we discuss the precise relationship between the constrained ℓ_1 -regularization problem (1.4) and the unconstrained ℓ_1 -regularization problem (1.1). In Section 2.4 we discuss properties of ℓ_1 -regularization in greater depth. Finally, in Section 2.5, we describe the idea of an orthant-wise projected search, which is a strategy typically adopted by orthant-wise methods in the literature.

2.1 Convexity

The problem of interest in this thesis is given by (1.1) whose objective function is of the form $F(x) = f(x) + \lambda\|x\|_1$ for $x \in \mathbb{R}^n$. In this thesis, the function f will always be assumed to be convex. Thus, we need the following two definitions.

Definition 1 (Convex set). *A set $\mathcal{X} \subseteq \mathbb{R}^n$ is convex if and only if for any $\{x_1, x_2\} \subseteq \mathcal{X}$ and any $t \in (0, 1)$, the point of $tx_1 + (1 - t)x_2$ belongs to \mathcal{X} .*

Definition 2 (Convex function). *Let \mathcal{X} be a convex set. A function $h : \mathcal{X} \rightarrow \mathbb{R}$ is convex if and only if for any $\{x_1, x_2\} \subseteq \mathcal{X}$ and $t \in (0, 1)$ it holds that*

$$h(t \cdot x_1 + (1 - t) \cdot x_2) \leq t \cdot h(x_1) + (1 - t) \cdot h(x_2).$$

It is straightforward to verify that the ℓ_1 -regularization term $\lambda\|x\|_1$ is a convex function since for any $\{x_1, x_2\} \subseteq \mathbb{R}^n$ and $t \in (0, 1)$ it holds that

$$\lambda\|tx_1 + (1 - t)x_2\|_1 \leq \lambda t\|x_1\|_1 + \lambda(1 - t)\|x_2\|_1 = t\lambda\|x_1\|_1 + (1 - t)\lambda\|x_2\|_1,$$

where we used the triangle-inequality and $\lambda > 0$. Subsequently, we can conclude that $F(x)$ is also a convex function since the sum of two convex functions with positive weights results in a convex function [53, Proposition 1.1.5].

For many algorithms, a stronger notion than convexity is often assumed for the problem functions. A popular example of such a notion, and one that is relevant to the thesis, is the following definition of a strongly convex function (we use a definition that assumes that f is differentiable).

Definition 3 (Strongly convex function). *Let \mathcal{X} be a convex set. A differentiable function $h : \mathcal{X} \rightarrow \mathbb{R}$ is strongly convex if and only if there exists a positive parameter $\sigma > 0$ such that for any $\{x_1, x_2\} \subseteq \mathcal{X}$ it holds that*

$$(\nabla h(x_1) - \nabla h(x_2))^T(x_1 - x_2) \geq \sigma\|x_1 - x_2\|^2.$$

Note that the previous inequality is equivalent to the following inequality:

$$h(x_2) \geq h(x_1) + \nabla h(x_1)^T(x_2 - x_1) + \frac{\sigma}{2}\|x_2 - x_1\|^2.$$

In the case that h is twice continuously differentiable, it holds that h is strongly convex with parameter $\sigma > 0$ if and only if $\nabla^2 h(x) \succeq \sigma I$ for all $x \in \mathcal{X}$, i.e. that the matrix $\nabla^2 h(x) - \sigma I$ is positive semi-definite for all $x \in \mathcal{X}$.

2.2 Subdifferential and Subgradients

Let us begin with the definition of a subgradient of a function at a point, which is a generalization of the idea of a gradient for functions that are differentiable.

Definition 4 (Subgradient). *Let $h : \mathbb{R}^n \rightarrow \mathbb{R}$ be a convex function. We say that $g_h \in \mathbb{R}^n$ is a subgradient of h at x if and only if*

$$h(y) \geq h(x) + g_h^T(y - x) \text{ for all } y \in \mathbb{R}^n.$$

Definition 5 (Subdifferential). *Let $h : \mathbb{R}^n \rightarrow \mathbb{R}$ be a convex function. The set of all subgradients of h at $x \in \mathbb{R}^n$ is called the subdifferential of h at x . We will use $\partial h(x)$ to denote the subdifferential of h at x . As an example, for the convex function $h(x) = \|x\|_1$, the subdifferential at a point $x \in \mathbb{R}^n$ is given by*

$$\partial \|x\|_1 = \{g \in \mathbb{R}^n : g_i = 1 \text{ if } x_i > 0, g_i = -1 \text{ if } x_i < 0, g_i \in [-1, 1] \text{ if } x_i = 0.\}$$

We may now state the well-known necessary and sufficient optimal conditions for computing a global minimizer of problem (1.1).

Lemma 6. *Let $h : \mathbb{R}^n \rightarrow \mathbb{R}$ be a convex function. It follows that x^* is a global minimizer to the problem*

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ h(x) \tag{2.1}$$

if and only if

$$0 \in \partial h(x^*).$$

Proof. Let $x^* \in \mathbb{R}^n$. It holds that $0 \in \partial h(x^*)$ if and only if

$$h(y) \geq h(x^*) + 0^T(y - x^*) \text{ for all } y \in \mathbb{R}^n,$$

which is equivalent to

$$h(y) \geq h(x^*) \text{ for all } y \in \mathbb{R}^n,$$

which holds if and only if x^* is a global minimizer of h . □

From Lemma 6, it follows that the necessary and sufficient optimality condition for problem (1.1) is

$$0 \in \partial F(x) \equiv \nabla f(x) + \lambda \partial \|x\|_1. \quad (2.2)$$

The subdifferential of the sum of two functions is not always equal to the sum of the subdifferentials of each convex function. However, for the function F under consideration, it does hold (see [53, Proposition 5.4.6]), which we used to obtain (2.2). Motivated by (2.2), algorithms for solving (1.1) aim at generating a sequence of iterates such that limit points of the produced sequence satisfy (2.2).

Among the subgradients in $\partial F(x)$, one subgradient receives special attention, namely the minimum norm element. In particular, from the argument in [54], it follows that the steepest descent direction for a convex function is the negation of the minimum-norm element of the subdifferential.

Definition 7 (Subgradient with minimum norm). *Let $h : \mathbb{R}^n \rightarrow \mathbb{R}$ be a convex function. We use $\tilde{g}_h(x)$ to denote the minimum-norm element of the subdifferen-*

tial of h at x , i.e.,

$$\tilde{g}_h(x) := \operatorname{argmin}_{g \in \partial h(x)} \|g\|_2.$$

For the function $F(x) = f(x) + \lambda\|x\|_1$, it was shown in [17] that the minimum norm element of the subdifferential has the following closed form:

$$[\tilde{g}_F(x)]_i = \begin{cases} [\nabla f(x)]_i + \lambda & \text{if } [x]_i > 0, \text{ or } [x]_i = 0 \text{ and } [\nabla f(x)]_i < -\lambda, \\ 0 & \text{if } [x]_i = 0 \text{ and } [\nabla f(x)]_i \in [-\lambda, \lambda], \\ [\nabla f(x)]_i - \lambda & \text{if } [x]_i < 0, \text{ or } [x]_i = 0 \text{ and } [\nabla f(x)]_i > \lambda. \end{cases} \quad (2.3)$$

Since the minimum norm element of the subdifferential of F is easily computable, it may be used in the construction of algorithms.

2.3 Constrained and Unconstrained

ℓ_1 -Formulations

In Section 1.2, it was mentioned that problem (1.4) can be used to produce sparse solution vectors. It was also mentioned that this constrained optimization problem is related to an unconstrained problem that shares similar properties. The next result makes this precise. Note that for a differentiable function f (as is assumed throughout this thesis) it holds that $\partial f(x) = \nabla f(x)$ for all $x \in \mathbb{R}^n$, and thus we use $\partial f(x)$ in the proof to highlight that this result holds without having to assume that f is differentiable.

Theorem 8. *The following relationships hold between problems (1.1) and (1.4).*

(i) *If x^* is a solution to (1.4) with $t > 0$ and $y^* \in \mathbb{R}$ is an associated Lagrange multiplier, then x^* solves (1.1) when $\lambda \equiv y^*$.*

(ii) *If x^* is a solution to (1.1) for $\lambda > 0$, then x^* solves (1.4) when $t \equiv \|x^*\|_1$.*

Proof. To prove part (i), let x^* be a minimizer to problem (1.4) with $t > 0$. Then, note that $\|0\|_1 - t < 0$ since $t > 0$, which means that the vector 0 is strictly feasible for the constraint in (1.4), i.e., Slater's constraint qualification [53, page 169] holds. Combining this with the fact that f and $\|x\|_1 - t$ are both convex functions, it follows from [53, Proposition 5.3.2] that there exists a Lagrange multiplier y^* satisfying

$$\|x^*\|_1 - t \leq 0, \quad y^* \geq 0, \quad (\|x^*\|_1 - t)y^* = 0, \quad \text{and} \quad (2.4)$$

$$0 \in \partial(f(x^*) + y^*h(x^*)) = \partial f(x^*) + y^*\partial(\|x\|_1), \quad (2.5)$$

where we used [53, Proposition 5.4.6] and the fact that $\partial(\|x\|_1 - t) = \partial(\|x\|_1)$ to derive the last equality. It now follows from (2.5) and Lemma 6 that x^* is a solution to (1.1) if we choose $\lambda \equiv y^*$, as claimed.

Next, to prove part (ii), let x^* be a solution to (1.1) for some $\lambda > 0$. It follows from Lemma 6 and [53, Proposition 5.4.6] that

$$0 \in \partial(f(x^*) + \lambda\|x^*\|_1) = \partial f(x^*) + \lambda\partial(\|x^*\|_1). \quad (2.6)$$

If we now define $t = \|x^*\|_1$ then it also follows that

$$\|x^*\|_1 - t \leq 0, \quad \lambda > 0, \quad \text{and} \quad (\|x^*\|_1 - t)\lambda = 0.$$

By combining this with (2.6), we find that x^* solves (1.4) with Lagrange multiplier equal $y^* \equiv \lambda$ (compare these conditions to (2.4)), as claimed. \square

2.4 Properties of ℓ_1 -Regularization

In this section, we discuss the properties of ℓ_1 -regularization more deeply. Generally, ℓ_1 -regularization as used in (1.1) has two main properties: a sparsity-inducing property and a shrinkage property. The sparsity-inducing property encourages entries in the solutions to be zero. This property is often used to improve the interpretation of the resulting model because it selects variables that have the strongest effect on prediction accuracy of the resulting model. The shrinkage property shrinks the magnitude of variables so that the solution tends to have a smaller magnitude than the solution without regularization. This property is usually used to reduce the variance associated with the computed model, thus improving prediction accuracy on unseen data.

2.4.1 Sparsity-inducing property

The sparsity-inducing property of ℓ_1 -regularization can be described from a geometric perspective and one based on optimality conditions. Let us describe these two perspectives in this section.

For a geometric view, in Section 2.3 we showed that the ℓ_1 -regularization problem (1.1) is related to the constrained problem (1.4). The feasible set of problem (1.4) for a value $t \in (0, \infty)$ is $\{x : \|x\|_1 \leq t\}$. Such level sets associated with the ℓ_1 -norm are polytopes, and have vertices, edges, and faces at which one or more entries in x are zero. Moreover, the solution x^{ℓ_1} to the ℓ_1 -constrained problem must be located at the intersection of the feasible set and the lowest contour of f that intersects the feasible set nontrivially. Since the lowest contour of f that intersects the feasible region is likely to occur at such locations [8], it follows that the ℓ_1 -regularization promotes sparse solutions.

For comparison purposes, the ℓ_2 -regularization problem (1.3) is similarly

related to the constrained problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad \|x\|_2^2 \leq t^2. \quad (2.7)$$

A solution to this problem, denoted by x^{ℓ_2} , lies on the lowest level curve of f that intersects the feasible set $\{x : \|x\|_2 \leq t\}$. Since the feasible set $\{x : \|x\|_2 \leq t\}$ is a ball without vertexes, edges or faces, it is unlikely that x^{ℓ_2} will contain zero entries. Compared to ℓ_1 -regularization and ℓ_2 -regularization, solutions to the non-regularized problem (1.2), denoted as \hat{x} , is the centroid of the contours of $f(x)$ (see [8]), which rarely intersects an axis; thus components of \hat{x} are rarely zero. Figure 2.1 shows the geometry for x^{ℓ_1} , x^{ℓ_2} and \hat{x} in \mathbb{R}^2 . Observe that x^{ℓ_1} hits one vertex (one entry is zero), and that x^{ℓ_2} and \hat{x} have no zero entries.

Let us now consider a perspective based on optimality conditions. It follows from (2.2) that x^{ℓ_1} is a solution of (1.1) if and only if

$$0 \in \nabla f(x^{\ell_1}) + \lambda \partial \|x^{\ell_1}\|_1.$$

This inequality implies that if $|\nabla f(x^{\ell_1})_i| \leq \lambda$ for some i , then $[x^{\ell_1}]_i = 0$. This observation highlights that the size of λ controls the percentage of zero elements in a solution x^{ℓ_1} . (Of course, as λ changes, so does x^{ℓ_1} .) Larger values for λ leads to solutions with sparser solutions x^{ℓ_1} ; ℓ_2 -regularization does not have this property. As an extreme example, if λ is sufficiently large, then x^{ℓ_1} becomes exactly zero, while the components of x^{ℓ_2} approach zero as $\lambda \rightarrow \infty$, but rarely have any entries exactly equal to zero for finite values of λ .

Theorem 9. *Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be convex, differentiable, bounded below on the level set $\mathcal{L} = \{x \in \mathbb{R}^n : F(x) \leq F(x_0)\}$, and Lipschitz continuous over \mathcal{L} . If $x^{\ell_1}(\lambda)$ and $x^{\ell_2}(\lambda)$ denote solutions to problem (1.1) and problem (1.3) for a given $\lambda > 0$, respectively, then the following conditions hold:*

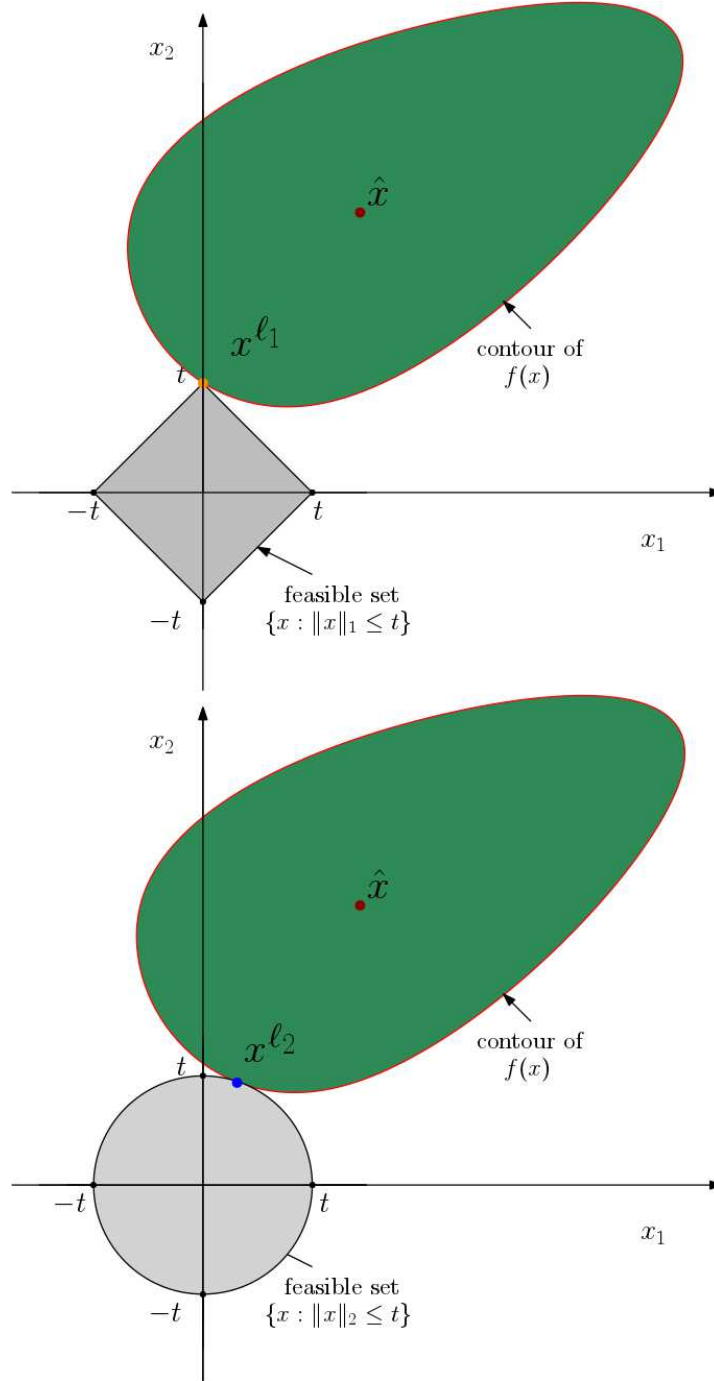


Figure 2.1: Geometry of ℓ_1 -regularization and ℓ_2 -regularization.

- (i) *There exists an $M > 0$ such that for any $\lambda > M$ it holds that $x^{\ell_1}(\lambda) = 0$.*
- (ii) *For each $i \in \{1, 2, \dots, n\}$, it holds that $[x^{\ell_2}(\lambda)]_i = 0$ if and only if $[\nabla f(x^{\ell_2}(\lambda))]_i = 0$. It follows that $x^{\ell_2}(\lambda) = 0$ if and only if $\nabla f(0) = 0$.*

(iii) It holds that $\lim_{\lambda \rightarrow \infty} x^{\ell_2}(\lambda) = 0$.

Proof. Since f is convex and Lipschitz continuous over \mathcal{L} , then it follows from [55, Lemma 2.6] that the gradient of f is uniformly bounded over \mathcal{L} . It follows that there exists $M < \infty$ such that

$$|[\nabla f(x)]_i| \leq M \text{ for all } x \in \mathcal{L} \text{ and all } i \in \{1, 2, \dots, n\}. \quad (2.8)$$

which will be used to prove parts (i)–(iii).

To prove part (i), it follows from the optimal condition (2.2) of problem (1.1) that for any $\lambda > M$ the ℓ_1 solution satisfies $x^{\ell_1}(\lambda) = 0$; otherwise there would exist some $i \in \{1, 2, \dots, n\}$ such that $[\nabla f(x^{\ell_1}(\lambda))]_i \in \{-\lambda, \lambda\}$, which is a contradiction since $\lambda > M \geq |[\nabla f(x^{\ell_1}(\lambda))]_i|$ because of (2.8).

To prove parts (ii) and (iii), first note that a solution $x^{\ell_2}(\lambda)$ to problem (1.3) must satisfy that the gradient is zero, i.e., that

$$\nabla f(x^{\ell_2}(\lambda)) + 2\lambda x^{\ell_2}(\lambda) = 0.$$

Since $\lambda > 0$ by assumption, it follows from the previous equality and (2.8) that

$$|[x^{\ell_2}(\lambda)]_i| = \frac{|[\nabla f(x^{\ell_2}(\lambda))]_i|}{2\lambda} \leq \frac{M}{2\lambda} \text{ for each } i \in \{1, 2, \dots, n\}. \quad (2.9)$$

We can now observe from the equality in (2.9) that part (ii) holds. Finally, by letting $\lambda \rightarrow \infty$ in (2.9), we also find that part (iii) holds. \square

2.4.2 Shrinkage property

The reason for the shrinkage property associated with ℓ_1 -regularization is analogous to the reason for ℓ_2 -regularization. Both ℓ_2 -regularization and ℓ_1 -regularization problems include penalties on the magnitude of variables, so that solutions

with large magnitude are discouraged. The size of λ controls the weight of this penalization so that larger values of λ is equivalent to heavier penalties, i.e., a greater shrinkage effect on the solution.

As a concrete example, consider the least-squares estimation function

$$f(x) = \|Ax - b\|_2^2, \quad (2.10)$$

where $A \in \mathbb{R}^{d \times n}$ is a real matrix with full column rank, and $b \in \mathbb{R}^d$. The minimizer of f defined in (2.10) is given by

$$\hat{x} = (A^T A)^{-1} A^T b \quad (2.11)$$

whereas the minimizer of $f(x) + \frac{\lambda}{2} \|x\|^2$ is given by

$$x^{\ell_2} = (A^T A + \lambda I)^{-1} A^T b. \quad (2.12)$$

On the other hand, when ℓ_1 -regularization is used, closed form solutions are generally not possible. Nonetheless, they can at least be approximated using some clever strategies. One popular approximation is motivated by the fact that the ℓ_1 penalty can be rewritten as follows:

$$\|x\|_1 = \sum_{i=1}^n |x_i| = \sum_{i=1}^n x_i^2 / |x_i|.$$

To compute an approximation, the denominator $|x_i|$ is regarded as fixed, so that it may be viewed as an ℓ_2 -penalty for which, as shown above, closed form solutions are possible. Specifically, given an approximation \tilde{x}^{ℓ_1} of x^{ℓ_1} , the solution of the ℓ_1 -regularization problem may be further approximated by

$$x^{\ell_1} \approx (A^T A + \lambda W^\dagger)^{-1} A^T b \quad (2.13)$$

where W is diagonal matrix whose i th diagonal entry is $||[\tilde{x}^{\ell_1}]_i||$, and W^\dagger is the generalized inverse of W . We finish this section by noting that it follows from (2.12) and (2.13) that for any $\lambda > 0$, the solutions x^{ℓ_1} and x^{ℓ_2} have smaller magnitude than \hat{x} , and that larger values for λ produce more shrinkage.

2.5 An Orthant-Wise Projected Search

For optimization algorithms, selecting an appropriate step size along directions computed to reduce the objective function is crucial for proving convergence. A popular method for finding suitable step sizes combines backtracking with the Armijo condition. The Armijo condition, which is defined next, is used to ensure that sufficient reduction along a given descent direction p_k is obtained.

Definition 10 (Armijo Condition). *Let $h : \mathbb{R}^n \rightarrow \mathbb{R}$ be a differentiable function and $\eta \in (0, 1)$. Given a point x_k and a descent direction p_k , we say that $\alpha \in (0, \infty)$ satisfies the Armijo condition if and only if*

$$h(x_k + \alpha p_k) \leq h(x_k) + \eta \alpha \nabla h(x)^T p_k.$$

For the framework that we propose, acceleration is obtained by performing subspace minimization based on predicting the orthant in which the solution is contained. For this reason, we must consider a more complicated linesearch than simply using a backtracking procedure until the Armijo condition above is satisfied. We describe this procedure next.

The core of our proposed search is the following orthant projection operator.

Definition 11 (Orthantwise Projection Operator). *Given $\bar{x} \in \mathbb{R}^n$, the orthant-*

wise projection of $x \in \mathbb{R}^n$, denoted by $\text{Proj}(x; \bar{x})$, is defined componentwise by

$$[\text{Proj}(x; \bar{x})]_i := \begin{cases} \max\{0, [x]_i\} & \text{if } [\bar{x}]_i > 0, \\ \min\{0, [x]_i\} & \text{if } [\bar{x}]_i < 0, \\ 0 & \text{if } [\bar{x}]_i = 0. \end{cases} \quad (2.14)$$

In the framework FaRSA that we propose, we must have a strategy for updating the k th iterate x_k . Our procedure will generate points y that may lie in a different orthant than x_k , so we will instead try $\text{Proj}(y; x_k)$ as prospective ways of defining x_{k+1} . Intuitively, this projection operator projects the trial point y onto the same orthant that x_k belongs.

Chapter 3

Related Work on ℓ_1 -Regularized Optimization Problems

In this chapter, we describe the most popular techniques developed to solve the ℓ_1 -regularization problem (1.1), and summarize their strengths and weaknesses. We start the description by reviewing proximal methods in Section 3.1, and then discuss the class of orthant-wise methods in Section 3.2. Finally, we summarize a two-metric projection method in Section 3.3.

3.1 Proximal Algorithms

The first optimization algorithms that we review belong to the class of *proximal algorithms*. Proximal algorithms are often used to solve structured non-smooth large-scale optimization problems. One especially popular problem is composite optimization, i.e., solving the problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) + h(x) \tag{3.1}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $h : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\}$ are closed proper convex functions, f is differentiable, and h may be non-differentiable, e.g. $h(x) = \|x\|_1$. The function h is usually referred to as the regularizer.

3.1.1 Proximal gradient method

During the k th iteration of the proximal gradient method, the function f is approximated by a first-order Taylor model, the regularizer is kept explicitly, and a proximal term relative to the current iterate x_k is added. Specifically, given the k th iterate x_k and proximal parameter $\alpha_k > 0$, the $(k+1)$ st iterate is

$$\begin{aligned} x_{k+1} &= \operatorname{argmin}_{x \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^T(x - x_k) + h(x) + \frac{1}{2\alpha_k} \|x - x_k\|_2^2 \\ &= \operatorname{argmin}_{x \in \mathbb{R}^n} \frac{1}{2\alpha_k} \|x - (x_k - \alpha_k \nabla f(x_k))\|_2^2 + h(x) \\ &=: \operatorname{Prox}_{\alpha_k h}(x_k - \alpha_k \nabla f(x_k)). \end{aligned} \tag{3.2}$$

The quadratic term, called the *proximal term*, penalizes deviation from x_k . For convergence the proximal parameter $\alpha_k > 0$ should be chosen less than two times the inverse of a local Lipschitz constant for ∇f at x_k . The proximal function, namely $\operatorname{Prox}_{\alpha_k h}$ is called the *proximal mapping* or *proximal operator*.

This thesis mainly focuses on the choice $h(x) = \lambda \|x\|_1$ for some given $\lambda > 0$. In the next two sections, we discuss this specific case in more detail.

3.1.1.1 Iterative Shrinkage-Thresholding Algorithm

If $h(x) = \lambda \|x\|_1$ for some $\lambda > 0$, then the proximal gradient iteration (3.2) has a closed form solution, namely

$$x_{k+1} = \operatorname{Prox}_{\alpha_k \lambda \| \cdot \|_1}(x_k - \alpha_k \nabla f(x_k)) \equiv \mathcal{S}(x_k - \alpha_k \nabla f(x_k); \lambda \alpha_k) \tag{3.3}$$

where \mathcal{S} is called the *shrinkage operator* in the literature. Specifically, each element of x_{k+1} is updated for $i \in \{1, 2, \dots, n\}$ as

$$[x_{k+1}]_i = \begin{cases} [x_k - \alpha_k \nabla f(x_k)]_i - \alpha_k \lambda & \text{if } [x_k - \alpha_k \nabla f(x_k)]_i > \alpha_k \lambda \\ [x_k - \alpha_k \nabla f(x_k)]_i + \alpha_k \lambda & \text{if } [x_k - \alpha_k \nabla f(x_k)]_i < -\alpha_k \lambda \\ 0 & \text{otherwise.} \end{cases} \quad (3.4)$$

Iteration (3.4) is the basis for the method ISTA (Iterative Shrinkage-Thresholding Algorithm). Under appropriate assumptions, e.g. $f(x)$ having a Lipschitz continuous gradient with Lipschitz constant L , ISTA can be shown to converge with rate $\mathcal{O}(1/k)$ when a fixed step size $\alpha_k = \alpha \in (0, 2/L)$ is used for all k [56]. If L is not known, the step sizes α_k may be found using a simple search technique [57, Chapter 2.4.3]. The $\mathcal{O}(1/k)$ is also known to be an upper complexity bound, meaning that the total number of iterations required by ISTA to obtain an ϵ -optimality solution is proportional to $1/\epsilon$. In the next section we discuss how acceleration techniques have been used in conjunction with ISTA.

3.1.1.2 Fast Iterative Shrinkage-Thresholding Algorithm

A few techniques exist for accelerating ISTA (see Section 3.1.1.1). The most well-known is often referred to as *Nesterov acceleration* [58]. Nesterov's acceleration strategy includes an extrapolation step during each iteration of the proximal gradient method. Specifically, the iteration is given by

$$\begin{aligned} y_{k+1} &= x_k + w_k(x_k - x_{k-1}) \\ x_{k+1} &= \text{Prox}_{\alpha_k h}(y_{k+1} - \alpha_k \nabla f(y_{k+1})) \end{aligned} \quad (3.5)$$

where $w_k \in [0, 1)$ is an extrapolation parameter. When $h(x) = \lambda \|x\|_1$ for some $\lambda > 0$, the algorithm is called FISTA (Fast Iterative Shrinkage-Thresholding

Algorithm) [20]. FISTA can achieve a convergence rate of $\mathcal{O}(1/k^2)$ with a fixed step size of $\alpha_k = \alpha \in (0, 2/L)$ for all k . Compared with ISTA, the upper complexity bound for FISTA is reduced from $\mathcal{O}(1/k)$ to $\mathcal{O}(1/k^2)$. In other words, the maximum number of iterations required by FISTA to obtain an ϵ -optimal solution is proportional to $1/\sqrt{\epsilon}$, which is better than the value $1/\epsilon$ for ISTA whenever $\epsilon \in (0, 1)$. This explains why the extrapolation technique introduced by Nesterov is referred to as acceleration.

3.1.2 Proximal Newton method

Both ISTA and FISTA (see Section 3.1.1.1 and 3.1.1.2) are first-order methods, i.e., they only use first-order derivatives of f . They have proved quite useful in practice because of their simplicity. Nonetheless, first-order methods are somewhat limited in terms of local convergence rate and robustness on ill-conditioned problems, which can often be overcome by employing second-order derivative information. To incorporate second-order derivative information into proximal methods, proximal Newton methods use the update

$$x_{k+1} = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2\alpha_k} (x - x_k)^T \nabla^2 f(x_k) (x - x_k) + h(x), \quad (3.6)$$

where x_k denotes the k th iterate and $\alpha_k > 0$ is the k th parameter. The difference between (3.2) and (3.6) is that the proximal Newton iteration utilizes the Hessian matrix of $f(x)$ at x_k . Since the class of proximal Newton methods uses second-derivative information, they are generally more computationally expensive per iteration when compared to proximal gradient methods. On the other hand, since proximal Newton methods use second-derivative curvature information, they often exhibit a dramatic reduction in the required number of iterations to reach a desired optimality tolerance when compared

with proximal gradient methods. For the ℓ_1 -regularization case, i.e. when $h(x) = \lambda\|x\|_1$ for some $\lambda > 0$, unlike the ISTA subproblem in (3.4), the proximal Newton subproblem in (3.6) generally does not have a closed-form solution. Therefore, one must apply an iterative method to compute an approximate solution x_{k+1} to (3.6). The most popular approach is to apply a coordinate descent (CD) algorithm to the subproblem to obtain an approximate solution x_{k+1} . For example, the state-of-the-art software LIBLINEAR, which implements newGLMNET [18], uses a CD algorithm to approximately minimize the piecewise quadratic subproblem. In practice, LIBLINEAR performs better than ISTA and FISTA in most cases, and represents a key competitor to the FaRSA framework that we propose.

3.2 Orthant Based Methods

Orthant based methods form another class of methods for solving problem (1.1). Such methods minimize a smooth quadratic model of $f(x)$ over a sequence of orthants of \mathbb{R}^n until a solution is found. Therefore, a key desirable feature of orthant methods is to efficiently and effectively identify the orthant containing a solution so that a correct optimal active set (the set of zero variables of a solution) is found. This is an attractive property since ℓ_1 -regularization tends to produce sparse solutions, meaning that optimal active sets are relatively large; equivalently, the number of nonzero variables is usually relatively small, which is a feature that our proposed method uses to its advantage. In the remainder of this section, we review two orthant methods: the Orthant-Wise Limited-memory Quasi-Newton Method (OWL-QN) [19] in Section 3.2.1 and the Orthant-Based Adaptive Method (OBA) [17] in Section 3.2.2.

3.2.1 Orthant-Wise Limited-Memory Quasi-Newton Method

The Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) method was one of the most effective method in experiments for ℓ_1 -regularization problems before the appearance of the improved proximal Newton method, e.g. newGLM-NET [18] in LIBLINEAR. The iteration for OWL-QN is based on selecting an appropriately defined descent direction for (1.1), choosing an orthant to be explored, computing a Newton-like direction within the orthant using an L-BFGS Hessian approximation, and reducing objective function along the computed Newton-like direction in the orthant being explored.

More specifically, the descent direction, denoted as v_k , is selected in OWL-QN as the negative of the minimum-norm element of the subdifferential of the objective $F(x) = f(x) + \lambda\|x\|_1$ (see (2.3)). OWL-QN then chooses an orthant face to be explored based on the current iterate x_k and the descent direction v_k . Specifically, the orthant face is defined as $\Omega_k = \{x \in \mathbb{R}^n : \text{Proj}(x; \xi_k) = x\}$ where $\text{Proj}(\cdot; \xi_k)$ is the orthantwise projection operator introduced in Definition 2.14 and $\xi_k \in \{-1, 0, +1\}^n$ is determined by x_k and v_k . By construction, it follows that for all $x \in \Omega_k$ the objective function is equal to

$$f(x) + \lambda \cdot \xi_k^T x$$

Next, OWL-QN approximates the objective function in Ω_k with a quadratic model, and then computes a Newton-like search direction by computing

$$d_k = H(x_k)^{-1} v_k \tag{3.7}$$

where $H(x_k)$ is the Hessian matrix of f at $x_k \in \Omega_k$. Although computing the exact inverse of the Hessian matrix $H(x_k)$ is expensive ($\mathcal{O}(n^3)$ time complexity

for an n -by- n matrix), an L-BFGS Hessian approximation is used by OWL-QN.

A line search mechanism is a natural choice for computing an appropriate step size to achieve sufficient reduction of the objective function, and also to ensure exploration within the desired orthant face Ω_k . However, a standard backtracking line search does not have a mechanism for remaining in the chosen orthant. To address this issue, the orthantwise projection operator (recall Definition 2.14) is used to project trial iterates onto the chosen orthant face. In addition, OWL-QN utilizes this projection operator to discard components of the search direction d_k if their signs are not consistent with those of v_k in order to ensure that d_k is a descent direction. Combining these ideas together, the iterate update of OWL-QN can be written as

$$x_{k+1} = \text{Proj}\{x_k + \alpha_k \text{Proj}\{d^k; v_k\}; \xi_k\} \quad (3.8)$$

where the step size α_k is computed to satisfy the Armijo condition but with the minimum norm element of the subdifferential v_k in place of the gradient.

Although OWL-QN is one of the better known methods, as pointed out in [17], on many test problems it is not competitive with other state-of-the-art solvers. In particular, OWL-QN has the following main disadvantages: (i) its orthant face selection is relative simple, which means that many iterates are often needed until the optimal orthant face is discovered; and (ii) the computation of the Newton-like step in (3.7) may be high since a *full* Hessian matrix approximation is used during each iteration.

3.2.2 Orthant-Based Adaptive Method

The Orthant-Based Adaptive (OBA) method is a more advanced orthant-based strategy [17]. Compared with OWL-QN, OBA has a more reliable orthant face

selection mechanism, as well as a scheme for controlling the subproblem size. An iteration of OBA consists of a corrective cycle for orthant face predictions and subspace minimization steps. The corrective cycle is terminated when the orthant-face prediction is deemed to be reliable. A globalization mechanism is used to determine if the computed trial iterate should be accepted or modified.

During their corrective cycle, an orthant face prediction is carried out by an active-set strategy. At the beginning of each iteration, an active-set estimation is produced based on the orthant face of x_k and an optimal measure. In particular, OBA defines a working set \mathcal{W}_k as

$$\mathcal{W}_k = \{i \mid [x_k]_i \neq 0, \text{ or } [x_k]_i = 0 \text{ and } |[\nabla f(x_k)]_i| > \lambda\},$$

which houses the indices of non-zero entries of x_k as well as zero entries of x_k that are deemed far from optimal. The complement of \mathcal{W}_k is denoted as \mathcal{A}_k whose variables are fixed in the following corrective cycle.

For each corrective cycle, first a search direction d_k that approximately minimizes a quadratic model of F over the variables in the working set is computed, e.g. using CG, so that the trial iterate becomes $\hat{x}_k = x_k + d_k$. An orthant-based learning strategy is then triggered, which involves determining which components of the trial iterate \hat{x}_k fall into a different orthant than x_k . Next the working set is updated by removing these violating elements, and then a new search direction and trial iterate are computed. This process iterates until no elements of the trial iterate fall into a different orthant than the one associated with x_k . Once a reliable orthant face is identified, a projected line search is applied to produce a new trial step. In order to ensure global convergence, OBA requires a globalization mechanism. This globalization mechanism computes a full ISTA step to further correct the trial step and to produce a new iterate.

Generally, OBA outperforms OWL-QN and is one of the current state-of-the-art solvers. Their orthant identification strategy usually efficiently identifies an optimal orthant but also helps control the subproblem size, which helps reduce the iteration cost. OBA is also flexible in the sense that various subproblem solvers may be used, e.g. the CG method, which helps OBA avoid potential issues with using a CD solver. As is discussed in [17], OBA usually performs better than LIBLINEAR on problems with a Hessian matrix that is not diagonally dominant. Finally, we conclude this section by pointing out two potential drawbacks of the OBA method: (i) the orthant face prediction during each iteration may be expensive since this prediction may require approximately minimizing a sequence of quadratic subproblems; and (ii) the convergence theory hinges on having to compute an ISTA step to gauge progress, which highlights that the strategy in its purest form is not guaranteed to converge.

3.3 Equivalent Smooth Reformulations

Another strategy for addressing ℓ_1 -norm minimization problems is to transform the non-smooth problem into a smooth problem. One such realization is to transform the elements of variables into its “positive” and “negative” parts, and then rewrite the ℓ_1 -norm as a sum of these parts. Specifically, for problem (1.1) we may write each x_i using auxiliary non-negative variables u_i and v_i satisfying $x_i = u_i - v_i$. Then, the unconstrained non-smooth ℓ_1 -regularization problem is equivalent to the following bounded-constrained and smooth problem

$$\underset{u, v \in \mathbb{R}^n}{\text{minimize}} \quad f(u - v) + \lambda e^T(u + v) \quad \text{subject to} \quad u, v \geq 0 \quad (3.9)$$

where $e = (1, 1, \dots, 1)^T \in \mathbb{R}^n$. The advantage of this smooth formulation is that it alleviates the challenges associated with nonsmooth optimization. However,

by introducing extra variables the computational footprint increases, although this may not be significant if the linear algebra is carefully organized.

There are many efficient methods available for solving smooth optimization problems with bound constraints [59, 60, 61, 62, 63, 64, 41, 38]. Here, we outline one such method that uses active-set techniques and is implemented in the ASA-CG software [38]. The ASA-CG algorithm consists of non-monotone gradient projection steps, unconstrained optimization steps, and a set of rules for deciding between the two steps. The specific implementation ASA-CG exploits a cyclic Barzilai Borwein (CBB) algorithm [65] for the gradient projection steps and a CG algorithm for the unconstrained optimization steps. The non-monotone gradient projection algorithm (NGPA) employs a non-monotone line search based on projected gradient calculations to arrive at a new iterate x_{k+1} . In essence, ASA-CG uses NGPA to identify active constraints, and uses the unconstrained optimization steps (e.g., the CG based iterations) to optimize f over the variables that are not deemed active by the NGPA. It is also important to remark that ASA-CG uses only first derivatives to approximate the Hessian matrix for its CG implementation, e.g. an L-BFGS approximation [66].

Chapter 4

FaRSA: A Solver for ℓ_1 -Regularized Optimization Problems

Even though there exist many approaches for solving ℓ_1 -regularization optimization problems, such as the state-of-arts solvers we presented in Chapter 3, they all have inherent drawbacks. Based on the description of those methods, we aim at designing a new algorithmic framework for solving ℓ_1 -regularization optimization problem that should involve the following main features: (i) It should be able to efficiently predict the correct orthant in which a solution is contained; (ii) It should be capable of using exact second-order derivative information in order to accelerate convergence; and (iii) It must be flexible so that future advancements (e.g. CG and CD strategies) may easily be incorporated.

In this chapter, we describe our *Fast Reduced Space Algorithm* (FaRSA) for solving (1.1). In Section 4.1, we first present an outline of FaRSA, and then describe the main components in detail. The convergence analysis of FaRSA is given in Section 4.2, including global and local convergence analysis results.

4.1 Algorithmic Framework

Crucial to our algorithm FaRSA is the manner in which we deal with the identification of the zero and nonzero elements of the solution. For predicting non-zero components, a step should be designed to free variables that are previously predicted to be zero. Once we obtain a prediction of the support (non-zero variables), we proceed to reduce the objective function in the reduced-space spanned by the predicted supports. Nevertheless, the predicted supports might be wrong, which means that FaRSA must be able to correct wrongly predicted variables. Therefore, an objective reducing step should be designed for two purposes: to decrease the objective function value in a reduced space, and to possibly correct variables that falsely predicted to be in the support. Besides, for the freeing and reducing steps, a set of rules is also needed by FaRSA to decide which of these steps to use during each iteration. One reasonable strategy is to consider optimality measures based on the spaces of free and fixed variables, i.e., if the best predicted reduction in f is best achieved by freeing variables (compute a freeing step) or reducing f in the space of currently free variables (compute a reducing step). An outline of FaRSA is presented as Algorithm 1.

Algorithm 1 Outline of FaRSA for solving problem (1.1).

```
1: Input:  $x_0$ 
2: for  $k = 0, 1, 2, \dots$  do
3:   if termination condition is satisfied then
4:     Return the (approximate) solution  $x_k$  of problem (1.1).
5:   if best predicted progress is to reduce  $f$  over current free variables then
6:     Do objective-reducing step.
7:   else
8:     Do variable-freeing step.
```

In the next sections, we discuss how to use optimality measure to design a rule to branch between the variable-freeing and objective-reducing steps in Section 4.1.1, and then present how variable-freeing and objective-reducing

steps work in detail in Section 4.1.2 and Section 4.1.3, respectively.

4.1.1 Optimality measures

Crucial to our algorithm is the manner in which we handle the zero and nonzero components of a solution estimate. In order to describe the details of our approach, we first define the index sets

$$\mathcal{I}^0(x) := \{i : [x]_i = 0\}, \quad \mathcal{I}^+(x) := \{i : [x]_i > 0\}, \quad \text{and} \quad \mathcal{I}^-(x) := \{i : [x]_i < 0\}.$$

We call $\mathcal{I}^0(x)$ the set of *zero variables*, $\mathcal{I}^+(x)$ the set of *positive variables*, $\mathcal{I}^-(x)$ the set of *negative variables*, and the union of $\mathcal{I}^-(x)$ and $\mathcal{I}^+(x)$ the set of *nonzero variables* at x . We use these sets to define measures of optimality corresponding to the zero and nonzero variables at x . Respectively, these measures are defined as follows:

$$[\beta(x)]_i := \begin{cases} \nabla_i f(x) + \lambda & \text{if } i \in \mathcal{I}^0(x) \text{ and } \nabla_i f(x) + \lambda < 0, \\ \nabla_i f(x) - \lambda & \text{if } i \in \mathcal{I}^0(x) \text{ and } \nabla_i f(x) - \lambda > 0, \\ 0 & \text{otherwise;} \end{cases}$$

$$[\phi(x)]_i := \begin{cases} 0 & \text{if } i \in \mathcal{I}^0(x), \\ \min\{\nabla_i f(x) + \lambda, \max\{[x]_i, \nabla_i f(x) - \lambda\}\} & \text{if } i \in \mathcal{I}^+(x) \text{ and } \nabla_i f(x) + \lambda > 0, \\ \max\{\nabla_i f(x) - \lambda, \min\{[x]_i, \nabla_i f(x) + \lambda\}\} & \text{if } i \in \mathcal{I}^-(x) \text{ and } \nabla_i f(x) - \lambda < 0, \\ \nabla_i f(x) + \lambda \cdot \text{sgn}([x]_i) & \text{otherwise.} \end{cases}$$

If x is a solution to problem (1.1), then for any $i \in \mathcal{I}^0(x)$ it holds from the optimality conditions for problem (1.1) that $|\nabla_i f(x)| \leq \lambda$. Therefore, the size of

$[\beta(x)]_i$ indicates how far that *zero* variable is from being optimal. In short, the size of the vector $\beta(x)$ is a measure of optimality for the zero variables, i.e., for those in $\mathcal{I}^0(x)$. On the other hand, for any $i \in \mathcal{I}^+(x) \cup \mathcal{I}^-(x)$ it holds from the optimality conditions that $\nabla_i f(x) + \text{sgn}([x]_i)\lambda = 0$. Therefore, the size of $[\phi(x)]_i$ indicates how far that *nonzero* variable is from being optimal, although we note that its definition also takes into account the distance the nonzero variable can move before becoming zero, i.e., before switching orthants. In short, the size of the vector $\phi(x)$ is a measure of optimality for the nonzero variables, i.e., for those in $\mathcal{I}^+(x) \cup \mathcal{I}^-(x)$. Note that the summation of $\phi(x)$ and $\beta(x)$ is the same as the negative of the well-known ISTA step, as we now show.

Lemma 12. *For any k , let s_k be the full ISTA step defined by*

$$s_k := \mathcal{S}(x_k - \nabla f(x_k); \lambda) - x_k, \text{ where}$$

$$[\mathcal{S}(x_k - \nabla f(x_k); \lambda) - x_k]_i := \begin{cases} -[\nabla f(x_k)]_i + \lambda & \text{if } [x_k - \nabla f(x_k)]_i < -\lambda, \\ -[x_k]_i & \text{if } [x_k - \nabla f(x_k)]_i \in [-\lambda, \lambda], \\ -[\nabla f(x_k)]_i - \lambda & \text{if } [x_k - \nabla f(x_k)]_i > \lambda, \end{cases}$$

and \mathcal{S} is defined in (3.3). Then, $s_k = -(\beta(x_k) + \phi(x_k))$.

Proof. Recall the definitions of the components of $\beta(x_k)$ and $\phi(x_k)$, which may

be rewritten in a slightly more convenient form as follows:

$$[\beta(x_k)]_i := \begin{cases} \nabla_i f(x_k) + \lambda & \text{if } [x_k]_i = 0 \text{ and } \nabla_i f(x_k) + \lambda < 0, \\ \nabla_i f(x_k) - \lambda & \text{if } [x_k]_i = 0 \text{ and } \nabla_i f(x_k) - \lambda > 0, \\ 0 & \text{otherwise,} \end{cases}$$

$$[\phi(x_k)]_i := \begin{cases} 0 & \text{if } [x_k]_i = 0, \\ \min\{\nabla_i f(x_k) + \lambda, \max\{[x_k]_i, \nabla_i f(x_k) - \lambda\}\} & \text{if } [x_k]_i > 0 \text{ and } \nabla_i f(x_k) + \lambda > 0, \\ \max\{\nabla_i f(x_k) - \lambda, \min\{[x_k]_i, \nabla_i f(x_k) + \lambda\}\} & \text{if } [x_k]_i < 0 \text{ and } \nabla_i f(x_k) - \lambda < 0, \\ \nabla_i f(x_k) + \lambda \cdot \text{sgn}([x_k]_i) & \text{otherwise.} \end{cases}$$

For any component i , we proceed by considering various cases and subcases.

Case 1: Suppose that

$$[x_k - \nabla f(x_k)]_i > \lambda, \text{ meaning that } [x_k]_i > \nabla_i f(x_k) + \lambda. \quad (4.1)$$

Subcase 1a: Suppose that $[x_k]_i > 0$ and $\nabla_i f(x_k) + \lambda > 0$, so $\nabla_i f(x_k) > -\lambda$. Then, $[\beta(x_k)]_i = 0$ and

$$[\phi(x_k)]_i = \min\{\nabla_i f(x_k) + \lambda, \max\{[x_k]_i, \nabla_i f(x_k) - \lambda\}\}. \quad (4.2)$$

By (4.1), it follows that $[x_k]_i > \nabla_i f(x_k) + \lambda > \nabla_i f(x_k) - \lambda$, which with $[x_k]_i > 0$ means the \max in (4.2) evaluates as $[x_k]_i$. Then, again with (4.1), the \min in (4.2) yields

$$[\phi(x_k)]_i = \nabla_i f(x_k) + \lambda = -[s_k]_i. \quad (4.3)$$

Subcase 1b: Suppose that $[x_k]_i > 0$ and $\nabla_i f(x_k) + \lambda \leq 0$, so $\nabla_i f(x_k) \leq -\lambda$. Then,

$[\beta(x_k)]_i = 0$ and

$$[\phi(x_k)]_i = \nabla_i f(x_k) + \lambda = -[s_k]_i. \quad (4.4)$$

Subcase 1c: Suppose that $[x_k]_i = 0$ and $\nabla_i f(x_k) + \lambda \leq 0$, so $\nabla_i f(x_k) \leq -\lambda$. Then,

$[\phi(x_k)]_i = 0$ and

$$[\beta(x_k)]_i = \nabla_i f(x_k) + \lambda = -[s_k]_i. \quad (4.5)$$

Subcase 1d: Suppose that $[x_k]_i < 0$ and $\nabla_i f(x_k) + \lambda < 0$, so $\nabla_i f(x_k) < -\lambda$. Then,

$[\beta(x_k)]_i = 0$ and

$$[\phi(x_k)]_i = \max\{\nabla_i f(x_k) - \lambda, \min\{[x_k]_i, \nabla_i f(x_k) + \lambda\}\}. \quad (4.6)$$

By (4.1), it follows that $[x_k]_i > \nabla_i f(x_k) + \lambda$, which along with $\nabla_i f(x_k) + \lambda < 0$ means that the min in (4.6) evaluates as $\nabla_i f(x_k) + \lambda$. Then, since $\nabla_i f(x_k) + \lambda > \nabla_i f(x_k) - \lambda$, the max in (4.6) yields

$$[\phi(x_k)]_i = \nabla_i f(x_k) + \lambda = -[s_k]_i. \quad (4.7)$$

Since Subcases 1a–1d exhaust all possibilities under (4.1), we conclude from (4.3), (4.4), (4.5), and (4.7) that for Case 1 we have $[s_k]_i = -[\beta(x_k) + \phi(x_k)]_i$.

Case 2: Suppose that

$$[x_k - \nabla f(x_k)]_i < -\lambda, \text{ meaning that } [x_k]_i < \nabla_i f(x_k) - \lambda. \quad (4.8)$$

We claim that the analysis for this case is symmetric to that in Case 1 above, from which we may conclude again that $[s_k]_i = -[\beta(x_k) + \phi(x_k)]_i$.

Case 3: Suppose that

$$[x_k - \nabla f(x_k)]_i \in [-\lambda, \lambda], \text{ meaning that } [x_k]_i \in \nabla_i f(x_k) + [-\lambda, \lambda]. \quad (4.9)$$

Subcase 3a: Suppose that $[x_k]_i > 0$. Then, $[\beta(x_k)]_i = 0$ and, since $[x_k]_i > 0$ and (4.9) imply $\nabla_i f(x_k) > -\lambda$,

$$[\phi(x_k)]_i = \min\{\nabla_i f(x_k) + \lambda, \max\{[x_k]_i, \nabla_i f(x_k) - \lambda\}\}. \quad (4.10)$$

Since (4.9) also implies $[x_k]_i > \nabla_i f(x_k) - \lambda$, it follows along with $[x_k]_i > 0$ that the max in (4.10) evaluates as $[x_k]_i$. Then, since (4.9) implies $[x_k]_i < \nabla_i f(x_k) + \lambda$, the min in (4.10) yields

$$[\phi(x_k)]_i = [x_k]_i = -[s_k]_i. \quad (4.11)$$

Subcase 3b: Suppose that $[x_k]_i = 0$. Then, $[\phi(x_k)]_i = 0$ and under (4.9), that

$$[\beta(x_k)]_i = -[s_k]_i = 0. \quad (4.12)$$

Subcase 3c: Suppose that $[x_k]_i < 0$. We claim that the analysis for this case is symmetric to that in Subcase 3.a, from which we may conclude that for this subcase we again have

$$[\phi(x_k)]_i = [x_k]_i = -[s_k]_i. \quad (4.13)$$

Since Sub-cases 1.a–1.d exhaust all possibilities under (4.9), we conclude from (4.11), (4.12), and (4.13) that for Case 3 we have $[s_k]_i = -[\beta(x_k) + \phi(x_k)]_i$. The result follows as we have proved the desired result under all cases. \square

The following result demonstrates that the functions β and ϕ together correspond to a valid optimality measure for problem (1.1).

Lemma 13. *Let S be an infinite set of positive integers such that $\{x_k\}_{k \in S} \rightarrow x_*$. Then, the x_* is an optimal solution to (1.1) if and only if $\{\beta(x_k)\}_{k \in S} \rightarrow 0$ and $\{\phi(x_k)\}_{k \in S} \rightarrow 0$. Consequently, x_* is an optimal solution to (1.1) if and only if $\|\beta(x_*)\| = \|\phi(x_*)\| = 0$.*

Proof. Suppose $\{\beta(x_k)\}_{k \in \mathcal{S}} \rightarrow 0$ and $\{\phi(x_k)\}_{k \in \mathcal{S}} \rightarrow 0$. Then, first, consider any i such that $[x_*]_i > 0$, which means that $[x_k]_i > 0$ for all sufficiently large $k \in \mathcal{S}$. We now consider two sub-cases. If $\nabla_i f(x_k) + \lambda \leq 0$ for infinitely many $k \in \mathcal{S}$, then it follows from the definition of $\phi(x_k)$, $\{\phi(x_k)\}_{k \in \mathcal{S}} \rightarrow 0$, and continuity of ∇f that $\nabla_i f(x_*) + \lambda = 0$. On the other hand, if $\nabla_i f(x_k) + \lambda > 0$ for infinitely many $k \in \mathcal{S}$, then it follows from the definition of $\phi(x_k)$, $\{\phi(x_k)\}_{k \in \mathcal{S}} \rightarrow 0$, $[x_*]_i > 0$, and continuity of ∇f that $\nabla_i f(x_*) + \lambda = 0$. By combining both cases, we have established that $\nabla_i f(x_*) + \lambda = 0$, so that the i th component satisfies the optimality conditions (2.2). A similar argument may be used for the case when one considers i such that $[x_*]_i < 0$ to show that $\nabla_i f(x_*) - \lambda = 0$.

It remains to consider i such that $[x_*]_i = 0$. We have four sub-cases to consider. First, if infinitely many $k \in \mathcal{S}$ satisfy $[x_k]_i = 0$ and $\nabla_i f(x_k) + \lambda < 0$, then it follows from the definition of $\beta(x_k)$, $\{\beta(x_k)\}_{k \in \mathcal{S}} \rightarrow 0$, and continuity of ∇f that $\nabla_i f(x_*) + \lambda = 0$; a similar argument shows that if infinitely many $k \in \mathcal{S}$ satisfy $[x_k]_i = 0$ and $\nabla_i f(x_k) - \lambda > 0$, then $\nabla_i f(x_*) - \lambda = 0$. Second, if infinitely many $k \in \mathcal{S}$ satisfy $[x_k]_i = 0$ and $|\nabla_i f(x_k)| < \lambda$, then, trivially, $|\nabla_i f(x_*)| \leq \lambda$. Third, if infinitely many $k \in \mathcal{S}$ satisfy $[x_k]_i > 0$ and $\nabla_i f(x_k) + \lambda \leq 0$, then it follows from the definition of $\phi(x_k)$, $\{\phi(x_k)\}_{k \in \mathcal{S}} \rightarrow 0$, and continuity of ∇f that $\nabla_i f(x_*) + \lambda = 0$; a similar argument shows that if infinitely many $k \in \mathcal{S}$ satisfy $[x_k]_i < 0$ and $\nabla_i f(x_k) - \lambda \geq 0$, then $\nabla_i f(x_*) - \lambda = 0$. Fourth, if infinitely many $k \in \mathcal{S}$ satisfy $[x_k]_i > 0$ and $\nabla_i f(x_k) + \lambda > 0$, then it follows from the definition of $\phi(x_k)$, $\{\phi(x_k)\}_{k \in \mathcal{S}} \rightarrow 0$, and continuity of ∇f that $|\nabla_i f(x_*)| \leq \lambda$; a similar argument shows that if infinitely many $k \in \mathcal{S}$ satisfy $[x_k]_i < 0$ and $\nabla_i f(x_k) - \lambda < 0$, then $|\nabla_i f(x_*)| \leq \lambda$. By combining these sub-cases, we conclude that $|\nabla_i f(x_*)| \leq \lambda$, so the i th component satisfies the optimality condition (2.2).

To prove the reverse implication, suppose that x_* is a solution to problem (1.1). If $[x_*]_i > 0$, then $[\beta(x_k)]_i = 0$ for all sufficiently large $k \in \mathcal{S}$ and

$\{[\phi(x_k)]_i\}_{k \in \mathcal{S}} \rightarrow 0$ since $\nabla_i f(x_*) + \lambda = 0$ and ∇f is continuous. If $[x_*]_i < 0$, then $[\beta(x_k)]_i = 0$ for all sufficiently large $k \in \mathcal{S}$ and $\{[\phi(x_k)]_i\}_{k \in \mathcal{S}} \rightarrow 0$ since $\nabla_i f(x_*) - \lambda = 0$ and ∇f is continuous. Finally, if $[x_*]_i = 0$, then $|\nabla_i f(x_*)| \leq \lambda$ and continuity of ∇f imply that $\{[\beta(x_k)]_i\}_{k \in \mathcal{S}} \rightarrow 0$ and $\{[\phi(x_k)]_i\}_{k \in \mathcal{S}} \rightarrow 0$. This completes the proof. \square

FaRSA computes a sequence of iterates $\{x_k\}$. During each iteration, the sets $\mathcal{I}^0(x_k)$, $\mathcal{I}^+(x_k)$, and $\mathcal{I}^-(x_k)$ are identified, which are used to define $\beta(x_k)$ and $\phi(x_k)$. Following the justification of Lemma 13, when both $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$ are less than a prescribed tolerance $\epsilon > 0$, FaRSA can return x_k as an approximate solution to (1.1). Otherwise, it proceeds in one of two ways depending on the relative sizes of $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$. Since $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$ are used as optimality measure for zero and non-zero variables respectively, when $\|\beta(x_k)\| > \|\phi(x_k)\|$, progress toward optimality is best achieved by freeing at least one variable that is currently set to zero; otherwise, progress toward optimality is best achieved by reducing the objective function value in the reduced space spanned by all non-zero variables. At this point, Algorithm 1 can be enriched to become Algorithm 2.

Algorithm 2 Refined Outline of FaRSA for solving problem (1.1).

- 1: **Input:** x_0
 - 2: **for** $k = 0, 1, 2, \dots$ **do**
 - 3: **if** $\max\{\|\beta(x_k)\|, \|\phi(x_k)\|\} \leq \epsilon$ **then**
 - 4: **Return** the (approximate) solution x_k of problem (1.1).
 - 5: **if** $\|\beta(x_k)\| \leq \|\phi(x_k)\|$ **then**
 - 6: Do objective-reducing step.
 - 7: **else**
 - 8: Do variable-freeing step.
-

4.1.2 Variable-freeing step

When $\|\beta(x_k)\| > \|\phi(x_k)\|$, progress toward optimality is best achieved by freeing at least one variable that is currently set to zero, i.e. a variable-freeing step should be computed. We now state our proposed variable-freeing step of FaRSA as Algorithm 3. The selection of variables to be freed is based on the magnitude of the components of $\beta(x_k)$, and denote the set of selected variables as \mathcal{I}_k . If $i \in \mathcal{I}_k$, then $[\beta(x_k)]_i \neq 0$, which in turn means that $i \in \mathcal{I}^0(x_k)$, i.e., the i th variable has the value zero. The components of $\beta(x_k)$ that correspond to \mathcal{I}_k are then used to define the search direction $[d_k]_{\mathcal{I}_k} := -[\beta(x_k)]_{\mathcal{I}_k}$. With the search direction d_k computed, a standard backtracking Armijo line search is then applied to get an appropriate step size with sufficient decrease to update x_{k+1} . It follows from Lemma 12 that $-\beta(x_k)$ is equivalent to an ISTA step in the space of zero variables. If a unit step length is taken, i.e., if $x_{k+1} = x_k + d_k$, then x_{k+1} can be interpreted as the iterate that would be obtained by taking a *reduced ISTA step* in the space of variables indexed by \mathcal{I}_k .

Algorithm 3 Outline of variable-freeing step in FaRSA.

- 1: **if** $\|\beta(x_k)\| > \|\phi(x_k)\|$ **then**
 - 2: Set $\mathcal{I}_k \leftarrow \{i : [\beta(x_k)]_i \neq 0\}$.
 - 3: Set $[d_k]_{\mathcal{I}_k} \leftarrow -[\beta(x_k)]_{\mathcal{I}_k}$ and $[d_k]_i \leftarrow 0$ for $i \notin \mathcal{I}_k$.
 - 4: Use Algorithm 4 to compute $x_{k+1} \leftarrow \text{LINESEARCH}_{-\beta}(x_k, d_k, \eta, \xi)$.
-

Algorithm 4 A line search procedure for computing x_{k+1} for variable-freeing.

- 1: **procedure** $x_{k+1} = \text{LINESEARCH}_{-\beta}(x_k, d_k, \eta, \xi)$
 - 2: Set $j \leftarrow 0$ and $y_0 \leftarrow x_k + d_k$.
 - 3: **while** $F(y_j) > F(x_k) - \eta \xi^j \|d_k\|^2$ **do**
 - 4: Set $j \leftarrow j + 1$ and then $y_j \leftarrow x_k + \xi^j d_k$.
 - 5: **return** $x_{k+1} \leftarrow y_j$.
-

4.1.3 Objective-reducing step

The relationship $\|\beta(x_k)\| \leq \|\phi(x_k)\|$ indicates that significant progress toward optimality can still be achieved by reducing F over the current set of nonzero variables at x_k . Our proposed objective-reducing step is stated as Algorithm 5. The nonzero variables to be updated are chosen if $[\phi(x_k)]_i \neq 0$ which in turn means that $i \notin \mathcal{I}^0(x_k)$. All of the chosen nonzero variables form the set \mathcal{I}_k . To reduce F over the space of variables in \mathcal{I}_k , the algorithm makes use of a quadratic model of the objective function defined over the reduced-space to approximate objective function value at points near x_k , i.e.,

$$m_k(d) := F_k + g_k^T d + \frac{1}{2} d^T H_k d, \quad (4.14)$$

where $F_k \equiv F(x_k)$, and g_k and H_k represent the reduced gradient and Hessian of F defined over the free variables in \mathcal{I}_k at x_k , respectively.

Our analysis does not require an exact minimizer of m_k . Rather, we allow for the computation of any direction \bar{d}_k that satisfies the conditions $g_k^T \bar{d}_k \leq g_k^T d_k^R$ and $m_k(\bar{d}_k) \leq m_k(0)$, where the reference direction d_k^R is computed by minimizing m_k along the steepest decent direction as

$$d_k^R = -\alpha_k g_k, \quad \text{where } \alpha_k = \|g_k\|^2 / (g_k^T H_k g_k). \quad (4.15)$$

The first condition imposes how much descent is required by the search direction \bar{d}_k , while the second condition ensures that the model is reduced at least as much as a zero step. It will be shown (see Lemma 24) that the first condition guarantees that the decrease in F is bounded below by a positive constant factor of $\|g_k\|_2^2$ (see (4.32)). It will be shown (see Lemma 23) that the second condition ensures that \bar{d}_k is bounded by a multiple of $\|g_k\|$. Such conditions are sat-

isfied by a Newton step, by any CG iterate, and asymptotically by CD iterates. Once \bar{d}_k is obtained, the search direction d_k in the full space is obtained by filling its elements that correspond to the index set \mathcal{I}_k with the elements from \bar{d}_k , and setting the complementary set of variables to zero. With the search direction d_k computed, we call Algorithm 6 in line 7 of Algorithm 5, which performs a (non-standard) backtracking projected line search. This line search procedure makes use of the projection operator $\text{Proj}(\cdot; x_k)$ defined as (2.14). This operator projects vectors onto the orthant inhabited by x_k , a feature shared by OBA. The while-loop that starts in line 3 of Algorithm 6 checks whether the trial point y_j decreases the objective function F relative to its value at x_k when $\text{sgn}(y_j) \neq \text{sgn}(x_k)$. If the line search terminates in this while-loop, then this implies that at least one component of x_k that was nonzero has become zero for $x_{k+1} = y_j$. Since the dimension of the reduced space will therefore be decreased during the next iteration the procedure only requires $F(x_{k+1}) \leq F(x_k)$ instead of a more traditional sufficient decrease condition, e.g., one based on the Armijo condition. If line 7 of Algorithm 6 is reached, then the current trial iterate y_j satisfies $\text{sgn}(y_j) = \text{sgn}(x_k)$, i.e., the trial iterate has entered the same orthant as that inhabited by x_k . Once this has occurred, the method could then perform a standard backtracking Armijo line search as stipulated in the loop starting at line 12. For the purpose of guaranteeing convergence, however, the method first checks whether the largest step along d_k that stays in the same orthant as x_k (see lines 8 and 9) satisfies the Armijo sufficient decrease condition (see line 10). (This aspect makes our procedure different from a standard backtracking scheme.) If Algorithm 6 terminates in line 5 or 11, then at least one nonzero variable at x_k will have become zero at x_{k+1} . Otherwise, if Algorithm 6 terminates in line 14, then x_{k+1} and x_k are in the same orthant and sufficient decrease in F was achieved (i.e., the Armijo condition in line 13 was satisfied).

Algorithm 5 Outline of objective-reducing step in FaRSA.

- 1: **if** $\|\beta(x_k)\| \leq \|\phi(x_k)\|$ **then**
- 2: **Choose any** $\mathcal{I}_k \subseteq \{i : [\phi(x_k)]_i \neq 0\}$.
- 3: **Set** $H_k \leftarrow \nabla_{\mathcal{I}_k \mathcal{I}_k}^2 F(x_k)$ **and** $g_k \leftarrow \nabla_{\mathcal{I}_k} F(x_k)$.
- 4: **Compute the reference direction**

$$d_k^R \leftarrow -\alpha_k g_k, \quad \text{where } \alpha_k \leftarrow \|g_k\|^2 / (g_k^T H_k g_k).$$

- 5: **Compute any** $\bar{d}_k \approx \operatorname{argmin}_d m_k(d)$ **such that the following hold:**

$$g_k^T \bar{d}_k \leq g_k^T d_k^R \quad \text{and} \quad m_k(\bar{d}_k) \leq m_k(0).$$

- 6: **Set** $[d_k]_{\mathcal{I}_k} \leftarrow \bar{d}_k$ **and** $[d_k]_i \leftarrow 0$ **for** $i \notin \mathcal{I}_k$.
 - 7: **Use Algorithm 6 to compute** $x_{k+1} \leftarrow \text{LINESEARCH}_{\phi}(x_k, d_k, \mathcal{I}_k, \eta, \xi)$.
-

Algorithm 6 Line search procedure for computing x_{k+1} for objective-reducing.

- 1: **procedure** $x_{k+1} = \text{LINESEARCH}_{\phi}(x_k, d_k, \mathcal{I}_k, \eta, \xi)$
 - 2: **Set** $j \leftarrow 0$ **and** $y_0 \leftarrow \text{Proj}(x_k + d_k; x_k)$.
 - 3: **while** $\operatorname{sgn}(y_j) \neq \operatorname{sgn}(x_k)$ **do**
 - 4: **if** $F(y_j) \leq F(x_k)$ **then**
 - 5: **return** $x_{k+1} \leftarrow y_j$.
 - 6: **Set** $j \leftarrow j + 1$ **and then** $y_j \leftarrow \text{Proj}(x_k + \xi^j d_k; x_k)$.
 - 7: **if** $j \neq 0$ **then**
 - 8: **Set** $\alpha_B \leftarrow \operatorname{argsup} \{\alpha > 0 : \operatorname{sgn}(x_k + \alpha d_k) = \operatorname{sgn}(x_k)\}$.
 - 9: **Set** $y_j \leftarrow x_k + \alpha_B d_k$.
 - 10: **if** $F(y_j) \leq F(x_k) + \eta \alpha_B \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k}$ **then**
 - 11: **return** $x_{k+1} \leftarrow y_j$.
 - 12: **loop**
 - 13: **if** $F(y_j) \leq F(x_k) + \eta \xi^j \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k}$ **then**
 - 14: **return** $x_{k+1} \leftarrow y_j$.
 - 15: **Set** $j \leftarrow j + 1$ **and then** $y_j \leftarrow x_k + \xi^j d_k$.
-

4.1.4 Practical enhancements

To help meet scalability requirements, FaRSA is flexible enough to support many enhancements, such as the following strategies.

- **Weighted optimality measures:** Instead of using the sizes of $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$ to branch between the objective-reducing and variable-freeing steps,

the size of $\|\beta(x_k)\|$ and $\gamma\|\phi(x_k)\|$ are compared where the parameter $\gamma \in \mathbb{R}^+$.

- **Subproblem index set selection:** Rather than selecting all nonzero components of β (when a variable-freeing step is computed) and ϕ (when an objective-reducing step is computed), only a subset of them needs to be chosen. More specifically, a subset \mathcal{I}_k of variables are chosen such that the norm of $\phi(x_k)$ or $\beta(x_k)$ over that subset of variables is at least proportional to the norm of $\phi(x_k)$ or $\beta(x_k)$ over the full set of variables. This also allows control over the size of the subproblem, which may be as small as one-dimensional if so desired.

- **Super-linear convergence:** When the iterates are sufficiently close to the a solution, it is desirable to additional conditions such

$$\|H_k \bar{d}_k + g_k\| \leq \min\{\mu_k \|g_k\|^r, \|g_k\|^2\} \quad (4.16)$$

so that local super-linear convergence may hopefully be recovered. (The $\|g_k\|^2$ quantity appearing in the right-hand-side of (4.19) may be replaced by $\|\bar{d}_k\|^2$ without affecting the convergence theory that we present later, although a minor change to the proof of Lemma 33 is required. We prefer using the quantity $\|g_k\|^2$ because then only the left-hand-side changes while the subproblem is being solved to obtain \bar{d}_k .) Importantly, all of the required conditions on \bar{d}_k are satisfied by the exact solution to $H_k \bar{d}_k = -g_k$, and so are still satisfied asymptotically by a convergent iterative algorithm such as CG or CD.

4.1.5 The complete algorithm

By incorporating the practical enhancements from Section 4.1.4, we obtain the final version of FaRSA, which is formally stated as Algorithm 7. This version of FaRSA will be analyzed in Section 4.2.

Algorithm 7 FaRSA for solving problem (1.1).

```

1: Input:  $x_0$ 
2: Constants:  $\{\eta_\phi, \eta_\beta\} \subset (0, 1]$ ,  $\xi \in (0, 1)$ ,  $\eta \in (0, 1/2]$ , and  $\{\gamma, \epsilon\} \subset (0, \infty)$ 
3: for  $k = 0, 1, 2, \dots$  do
4:   if  $\max\{\|\beta(x_k)\|, \|\phi(x_k)\|\} \leq \epsilon$  then
5:     Return the (approximate) solution  $x_k$  of problem (1.1).
6:   if  $\|\beta(x_k)\| \leq \gamma\|\phi(x_k)\|$  then
7:     Choose any  $\mathcal{I}_k \subseteq \{i : [\phi(x_k)]_i \neq 0\}$  so that  $\|[\phi(x_k)]_{\mathcal{I}_k}\| \geq \eta_\phi\|\phi(x_k)\|$ .
8:     Set  $H_k \leftarrow \nabla_{\mathcal{I}_k \mathcal{I}_k}^2 F(x_k)$  and  $g_k \leftarrow \nabla_{\mathcal{I}_k} F(x_k)$ .
9:     Compute the reference direction

$$d_k^R \leftarrow -\alpha_k g_k, \text{ where } \alpha_k \leftarrow \|g_k\|^2 / (g_k^T H_k g_k).$$

10:    Compute any  $\bar{d}_k \approx \operatorname{argmin}_d m_k(d)$  such that the following hold:

$$g_k^T \bar{d}_k \leq g_k^T d_k^R, \tag{4.17}$$


$$m_k(\bar{d}_k) \leq m_k(0), \text{ and} \tag{4.18}$$


$$\|H_k \bar{d}_k + g_k\| \leq \min\{\mu_k \|g_k\|^r, \|g_k\|^2\}. \tag{4.19}$$

11:    Set  $[d_k]_{\mathcal{I}_k} \leftarrow \bar{d}_k$  and  $[d_k]_i \leftarrow 0$  for  $i \notin \mathcal{I}_k$ .
12:    Use Algorithm 6 to compute  $x_{k+1} \leftarrow \text{LINESEARCH}_\phi(x_k, d_k, \mathcal{I}_k, \eta, \xi)$ .
13:  else
14:    Choose any  $\mathcal{I}_k \subseteq \{i : [\beta(x_k)]_i \neq 0\}$  so that  $\|[\beta(x_k)]_{\mathcal{I}_k}\| \geq \eta_\beta\|\beta(x_k)\|$ .
15:    Set  $[d_k]_{\mathcal{I}_k} \leftarrow -[\beta(x_k)]_{\mathcal{I}_k}$  and  $[d_k]_i \leftarrow 0$  for  $i \notin \mathcal{I}_k$ .
16:    Use Algorithm 4 to compute  $x_{k+1} \leftarrow \text{LINESEARCH}_\beta(x_k, d_k, \eta, \xi)$ .

```

4.2 Convergence Analysis

In this section, we analyze the global and local convergence properties of FaRSA as presented as Algorithm 7. Our analysis uses the following assumption that is assumed to hold throughout this section.

Assumption 14. *The function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex, twice continuously differentiable, and bounded below on the level set $\mathcal{L} := \{x \in \mathbb{R}^n : F(x) \leq F(x_0)\}$. The gradient function $\nabla f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is Lipschitz continuous on \mathcal{L} with Lipschitz constant L . The Hessian $H \equiv \nabla^2 f : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ is uniformly positive definite*

and bounded on \mathcal{L} , i.e., there exist positive constants θ_{\min} and θ_{\max} so that

$$\theta_{\min}\|v\|^2 \leq v^T H(x)v \leq \theta_{\max}\|v\|^2 \text{ for all } \{x, v\} \subset \mathbb{R}^n.$$

The following remark concerning Assumption 14 is important.

Remark 15. *The assumption that the Hessian function $\nabla^2 f$ is uniformly positive definite and bounded on \mathcal{L} in Assumption 14 has been made for simplicity. Such an assumption can easily be replaced by the requirement that the sequence of matrices $\{H_k\}$ used in line 8 of Algorithm 7 be chosen as any symmetric positive-definite matrices with eigenvalues uniformly bounded above and away from zero. In this case, the global convergence and active set identification results presented later for Algorithm 7 also apply when f is merely convex; the local superlinear convergence results no longer apply.*

Our local convergence analysis requires the following additional assumption on the Hessian function in a neighborhood about x_* .

Assumption 16. *The Hessian function $\nabla^2 f : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ is Lipschitz continuous in a neighborhood of x_* with Lipschitz constant L_H .*

Being a reduced space method, FaRSA relies on determining the zero and nonzero components of the unique solution x_* . It follows from the optimality conditions for (1.1) that $|\nabla_i f(x_*)| \leq \lambda$ for all $i \in \mathcal{I}^0(x_*)$. Our local analysis needs x_* to satisfy the following stronger condition.

Assumption 17. *At the solution x_* , one finds $|\nabla_i f(x_*)| < \lambda$ for all $i \in \mathcal{I}^0(x_*)$.*

Assumption 17 is a strict complementarity assumption. In particular, if the smooth reformulation of problem (1.1) based on minimizing $f(u - v) + \lambda e^T(u + v)$ subject to the constraints $(u, v) \geq 0$ is used, then Assumption 17 is equivalent

to assuming that the Lagrange multipliers (z_u, z_v) associated with $(u, v) \geq 0$, which must be nonnegative, must satisfy $([z_v]_i, [z_u]_i) > 0$ for all $i \in \mathcal{I}^0(x_*)$.

4.2.1 Global convergence analysis

Our analysis uses the following index sets defined based on Algorithm 7:

$$\begin{aligned}\mathcal{S}_\phi &:= \{k : \text{lines 7--12 in Algorithm 7 are performed during iteration } k\}, \\ \mathcal{S}_\phi^{ADD} &:= \{k \in \mathcal{S}_\phi : \text{sgn}(x_{k+1}) \neq \text{sgn}(x_k)\}, \\ \mathcal{S}_\phi^{SD} &:= \{k \in \mathcal{S}_\phi : \text{sgn}(x_{k+1}) = \text{sgn}(x_k)\}, \text{ and} \\ \mathcal{S}_\beta &:= \{k : \text{lines 14--16 in Algorithm 7 are performed during iteration } k\}.\end{aligned}$$

We start with a lemma that gives an important identity for iterations in \mathcal{S}_β .

Lemma 18. *If $k \in \mathcal{S}_\beta$, then (\mathcal{I}_k, d_k) in lines 14 and 15 of Algorithm 7 yield*

$$[d_k]_{\mathcal{I}_k} = -[\nabla f(x_k) + \lambda \cdot \text{sgn}(x_k + \xi^j d_k)]_{\mathcal{I}_k} \text{ for any integer } j. \quad (4.20)$$

Thus, the right-hand side of (4.20) has the same value for any integer j .

Proof. We prove that (4.20) holds for any element of \mathcal{I}_k . To this end, let j be any integer and $i \in \mathcal{I}_k \subseteq \{\ell : [\beta(x_k)]_\ell \neq 0\}$, where \mathcal{I}_k is defined in line 14. It follows from definition of \mathcal{I}_k , definition of d_k in line 15, and $i \in \mathcal{I}_k$ that

$$[d_k]_i = \begin{cases} -(\nabla_i f(x_k) + \lambda) & \text{if } \nabla_i f(x_k) + \lambda < 0, \\ -(\nabla_i f(x_k) - \lambda) & \text{if } \nabla_i f(x_k) - \lambda > 0, \end{cases} \quad (4.21)$$

so that $[d_k]_i \neq 0$. Also, since $[x_k]_i = 0$ for $i \in \mathcal{I}_k$, we know that $[x_k + \xi^j d_k]_i \neq 0$.

Thus, we need only consider the following two cases.

Case 1: Suppose $[x_k + \xi^j d_k]_i > 0$. In this case, the right-hand-side of (4.20)

is equal to $-(\nabla_i f(x_k) + \lambda)$. As for the left-hand-side, since $[x_k]_i = 0$ and $[x_k + \xi^j d_k]_i > 0$, we have $0 < [x_k + \xi^j d_k]_i = \xi^j [d_k]_i$, which combined with $\xi^j > 0$ means that $[d_k]_i > 0$. Combining this fact with (4.21) gives $[d_k]_i = -(\nabla_i f(x_k) + \lambda)$, so that (4.20) holds.

Case 2: Suppose $[x_k + \xi^j d_k]_i < 0$. In this case, the right-hand-side of (4.20) is equal to $-(\nabla_i f(x_k) - \lambda)$. For the left-hand-side, since $[x_k]_i = 0$ and $[x_k + \xi^j d_k]_i < 0$ we have $0 > [x_k + \xi^j d_k]_i = \xi^j [d_k]_i$, which combined with $\xi^j > 0$ means that $[d_k]_i < 0$. This fact and (4.21) gives $[d_k]_i = -(\nabla_i f(x_k) - \lambda)$, so that (4.20) holds. \square

We can now establish a bound for a decrease in the objective when $k \in \mathcal{S}_\beta$.

Lemma 19. *If $k \in \mathcal{S}_\beta$, then d_k in line 15 of Algorithm 7 yields*

$$F(x_k + \xi^j d_k) \leq F(x_k) - \frac{\xi^j}{2} \|d_k\|^2 \text{ for any integer } j \text{ with } 0 \leq \xi^j \leq \frac{1}{L}.$$

Proof. Let j be any integer with $0 \leq \xi^j \leq \frac{1}{L}$ and let $y_j := x_k + \xi^j d_k$. By Lipschitz continuity of the gradient function ∇f , we have

$$\begin{aligned} f(y_j) &\leq f(x_k) + \xi^j \nabla f(x_k)^T d_k + \frac{L}{2} \|\xi^j d_k\|^2 \\ &\leq f(x_k) + \xi^j \nabla f(x_k)^T d_k + \frac{\xi^j}{2} \|d_k\|^2. \end{aligned} \tag{4.22}$$

It then follows from (4.22), convexity of both f and $\lambda \|\cdot\|_1$, the fact that $\text{sgn}(y_j) \in \partial \|y_j\|_1$, the definition of d_k (in particular that $[d_k]_i = 0$ for $i \notin \mathcal{I}_k$), and Lemma 18

that the following holds for all $z \in \mathbb{R}^n$:

$$\begin{aligned}
& F(y_j) \tag{4.23} \\
&= f(y_j) + \lambda \|y_j\|_1 \\
&\leq f(x_k) + \xi^j \nabla f(x_k)^T d_k + \frac{\xi^j}{2} \|d_k\|^2 + \lambda \|y_j\|_1 \\
&\leq f(z) + \nabla f(x_k)^T (x_k - z) + \xi^j \nabla f(x_k)^T d_k + \frac{\xi^j}{2} \|d_k\|^2 + \lambda \|z\|_1 + \lambda \cdot \text{sgn}(y_j)^T (y_j - z) \\
&\leq F(z) + [\nabla f(x_k) + \lambda \cdot \text{sgn}(y_j)]^T (x_k - z) + \xi^j [\nabla f(x_k) + \lambda \cdot \text{sgn}(y_j)]^T d_k + \frac{\xi^j}{2} \|d_k\|^2 \\
&= F(z) + [\nabla f(x_k) + \lambda \cdot \text{sgn}(y_j)]^T (x_k - z) - \xi^j \|d_k\|^2 + \frac{\xi^j}{2} \|d_k\|^2 \\
&= F(z) + [\nabla f(x_k) + \lambda \cdot \text{sgn}(y_j)]^T (x_k - z) - \frac{\xi^j}{2} \|d_k\|^2.
\end{aligned}$$

The desired result follows by considering $z = x_k$ in (4.23). \square

We now show that Algorithm 4 called in line 16 of Algorithm 7 is well defined, and that it returns x_{k+1} yielding sufficient decrease in the objective F .

Lemma 20. *If $k \in \mathcal{S}_\beta$, then x_{k+1} satisfies*

$$F(x_{k+1}) \leq F(x_k) - \kappa_\beta \max\{\|\beta(x_k)\|^2, \gamma^2 \|\phi(x_k)\|^2\}, \tag{4.24}$$

where $\kappa_\beta := \eta \eta_\beta \min\{1, \xi/L\}$.

Proof. Let j be any integer with $0 \leq \xi^j \leq \frac{1}{L}$ and let $y_j := x_k + \xi^j d_k$. It follows from Lemma 19 and the fact that $\eta \in (0, 1/2]$ in Algorithm 7 that

$$F(y_j) \leq F(x_k) - \frac{\xi^j}{2} \|d_k\|^2 \leq F(x_k) - \eta \xi^j \|d_k\|^2,$$

It follows from this inequality that Algorithm 4 will return the vector $x_{k+1} = x_k + \xi^{j^*} d_k$ with $\xi^{j^*} \geq \min\{1, \xi/L\}$ when called in line 16 of Algorithm 7. Using

this bound, line 3 of Algorithm 4, and lines 15 and 14 of Algorithm 7, we have

$$\begin{aligned}
F(x_{k+1}) &\leq F(x_k) - \eta \xi^{j^*} \|d_k\|^2 \leq F(x_k) - \eta \min\{1, \xi/L\} \|d_k\|^2 \\
&= F(x_k) - \eta \min\{1, \xi/L\} \|[\beta(x_k)]_{\mathcal{I}_k}\|^2 \\
&\leq F(x_k) - \eta \eta_\beta \min\{1, \xi/L\} \|\beta(x_k)\|^2.
\end{aligned}$$

The inequality (4.24) follows from the definition of κ_β , the previous inequality, and the fact that the inequality in line 6 of Algorithm 7 must not hold since line 16 is assumed to be reached. \square

We now show that the index set \mathcal{S}_β must be finite.

Lemma 21. *The index set \mathcal{S}_β must be finite, i.e., $|\mathcal{S}_\beta| < \infty$.*

Proof. To derive a contradiction, suppose that $|\mathcal{S}_\beta| = \infty$, which also means that Algorithm 7 does not terminate finitely. Since Algorithm 7 does not terminate finitely, we know from line 4 of Algorithm 7 that $\max\{\|\beta(x_k)\|, \|\phi(x_k)\|\} > \epsilon$ for all $k \geq 0$. Combining this inequality with Lemma 20 and the fact that $F(x_{k+1}) \leq F(x_k)$ for all $k \notin \mathcal{S}_\beta$ (as a result of Algorithm 6 called in line 12 of Algorithm 7), we may conclude for any nonnegative integer ℓ and $\kappa_\beta > 0$ defined in Lemma 20 that

$$\begin{aligned}
F(x_0) - F(x_{\ell+1}) &= \sum_{k=0}^{\ell} [F(x_k) - F(x_{k+1})] \\
&\geq \sum_{k \in \mathcal{S}_\beta, k \leq \ell} [F(x_k) - F(x_{k+1})] \\
&\geq \sum_{k \in \mathcal{S}_\beta, k \leq \ell} \kappa_\beta \max\{\|\beta(x_k)\|^2, \gamma^2 \|\phi(x_k)\|^2\} \\
&\geq \sum_{k \in \mathcal{S}_\beta, k \leq \ell} \kappa_\beta \min\{1, \gamma^2\} \epsilon^2.
\end{aligned}$$

Rearranging the previous inequality shows that

$$\begin{aligned}\lim_{l \rightarrow \infty} F(x_{\ell+1}) &\leq \lim_{\ell \rightarrow \infty} \left[F(x_0) - \sum_{k \in \mathcal{S}_\beta, k \leq \ell} \kappa_\beta \min\{1, \gamma^2\} \epsilon^2 \right] \\ &= F(x_0) - \sum_{k \in \mathcal{S}_\beta} \kappa_\beta \min\{1, \gamma^2\} \epsilon^2 = -\infty,\end{aligned}$$

which contradicts Assumption 14. Thus, we conclude that $|\mathcal{S}_\beta| < \infty$. \square

To prove that Algorithm 7 terminates finitely with an approximate solution to problem (1.1), all that remains is to prove that the set \mathcal{S}_ϕ is finite. To establish that $\mathcal{S}_\phi \equiv \mathcal{S}_\phi^{ADD} \cup \mathcal{S}_\phi^{SD}$ is finite, we proceed by showing individually that both \mathcal{S}_ϕ^{ADD} and \mathcal{S}_ϕ^{SD} are finite. We begin with the set \mathcal{S}_ϕ^{ADD} .

Lemma 22. *The set \mathcal{S}_ϕ^{ADD} is finite, i.e., $|\mathcal{S}_\phi^{ADD}| < \infty$.*

Proof. To derive a contradiction, suppose that $|\mathcal{S}_\phi^{ADD}| = \infty$, which in particular means that Algorithm 7 does not terminate finitely. Since Lemma 21 shows that \mathcal{S}_β is finite, we may also conclude that there exists an iteration k_1 such that $k \in \mathcal{S}_\phi = \mathcal{S}_\phi^{ADD} \cup \mathcal{S}_\phi^{SD}$ for all $k \geq k_1$.

We proceed by making two observations. First, if the i th component of x_k becomes zero for some iteration $k \geq k_1$, it will remain zero for the remainder of the iterations. This can be seen by using lines 11 and 7 of Algorithm 7 and the definition of $\phi(x_k)$ to deduce that if $[d_k]_i \neq 0$, then $i \in \mathcal{I}_k \subseteq \{\ell : [\phi(x_k)]_\ell \neq 0\} \subseteq \mathcal{I}^+(x_k) \cup \mathcal{I}^-(x_k)$ for all $k \geq k_1$; equivalently, if $i \in \mathcal{I}^0(x_k)$, then $[d_k]_i = 0$. The second observation is that at least one nonzero component of x_k becomes zero at x_{k+1} for each $k \in \mathcal{S}_\phi^{ADD}$. This can be seen by construction of Algorithm 6 when it is called in line 12 of Algorithm 7. Together, these observations contradict $|\mathcal{S}_\phi^{ADD}| = \infty$, since at most n variables may become zero. Thus, we must conclude that $|\mathcal{S}_\phi^{ADD}| < \infty$. \square

To establish that \mathcal{S}_ϕ^{SD} is finite, we require the following two lemmas. The first lemma gives a bound on the size of \bar{d}_k that holds whenever $k \in \mathcal{S}_\phi$.

Lemma 23. *If $k \in \mathcal{S}_\phi$, then $\|\bar{d}_k\| \leq (2/\theta_{\min})\|g_k\|$ where the constant $\theta_{\min} > 0$ is defined in Assumption 14.*

Proof. Let $k \in \mathcal{S}_\phi$ so that \bar{d}_k is computed in line 10 of Algorithm 7, and let d_k^N be the Newton step satisfying $H_k d_k^N = -g_k$ with H_k and g_k defined in line 8 of Algorithm 7. It follows that

$$\|d_k^N\| \leq \|H_k^{-1}\| \|g_k\|. \quad (4.25)$$

Let us also define the quadratic function $\bar{m}_k(d) := m_k(d_k^N + d)$ and the associated level set $\mathcal{L}_k := \{d : \bar{m}_k(d) \leq 0\}$. We then see that

$$(\bar{d}_k - d_k^N) \in \mathcal{L}_k \quad (4.26)$$

since $\bar{m}_k(\bar{d}_k - d_k^N) = m_k(\bar{d}_k) \leq m_k(0) = 0$, where we have used the condition $m_k(\bar{d}_k) \leq m_k(0)$ that is required to hold in line 10 of Algorithm 7.

We are now interested in finding a point in \mathcal{L}_k with largest norm. To characterize such a point, we consider the optimization problem

$$\underset{d \in \mathbb{R}^n}{\text{maximize}} \quad \frac{1}{2} \|d\|^2 \quad \text{subject to } d \in \mathcal{L}_k. \quad (4.27)$$

It is not difficult to prove that a global maximizer of problem (4.27) is $d_* := \alpha_* v$ with $\alpha_*^2 := (-g_k^T d_k^N)/\theta$, where (v, θ) with $\|v\| = 1$ is an eigenpair corresponding to the left-most eigenvalue $\theta \geq \theta_{\min}$ of H_k . (This can also be seen to hold since the level curves of \bar{m}_k are ellipses, $d = 0$ is the minimizer of \bar{m}_k , and the eigenvector corresponding to the left-most eigenvalue of H_k is the direction of least positive curvature.) Thus, it follows that $\|d\|^2 \leq \|d_*\|^2$ for all $d \in \mathcal{L}_k$. Combining this

with (4.26), the definition of d_* , and (4.25) shows that

$$\begin{aligned}\|\bar{d}_k - d_k^N\|^2 &\leq \|d_*\|^2 = \alpha_*^2 \|v\|^2 = \frac{-g_k^T d_k^N}{\theta} \\ &\leq \frac{\|g_k\| \|d_k^N\|}{\theta} \leq \frac{\|H_k^{-1}\| \|g_k\|^2}{\theta} = \left(\frac{\|g_k\|}{\theta}\right)^2.\end{aligned}$$

Combining this with the triangle inequality and (4.25), we obtain

$$\|\bar{d}_k\| \leq \|\bar{d}_k - d_k^N\| + \|d_k^N\| \leq \frac{\|g_k\|}{\theta} + \frac{\|g_k\|}{\theta} = \frac{2\|g_k\|}{\theta} \leq \frac{2\|g_k\|}{\theta_{\min}},$$

which complete the proof. □

The next result establishes a bound on the decrease in F when $k \in \mathcal{S}_\phi^{SD}$.

Lemma 24. *If $k \in \mathcal{S}_\phi^{SD}$, then x_{k+1} satisfies*

$$F(x_{k+1}) \leq F(x_k) - \kappa_\phi \max\{\gamma^{-2} \|\beta(x_k)\|^2, \|\phi(x_k)\|^2\}, \quad (4.28)$$

where $\kappa_\phi := \eta_\phi^2 \min\left\{\frac{\eta}{\theta_{\max}}, \frac{\eta\xi(1-\eta)\theta_{\min}^2}{2\theta_{\max}^3}\right\} > 0$.

Proof. Let $k \in \mathcal{S}_\phi^{SD}$. We consider two cases. First, suppose that $j = 0$ when line 7 in Algorithm 6 is reached. In this case, it follows by construction of Algorithm 6 that $\text{sgn}(y_0) = \text{sgn}(x_k + d_k) = \text{sgn}(x_k)$, i.e., the full step d_k and the vector x_k are contained in the same orthant. Consequently, the loop that starts in line 12 is simply a backtracking Armijo line search. Thus, if

$$\xi^j \in \left(0, \frac{2(\eta-1)\nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k}}{\theta_{\max} \|[d_k]_{\mathcal{I}_k}\|^2}\right] \equiv \left(0, \frac{2(\eta-1)g_k^T \bar{d}_k}{\theta_{\max} \|\bar{d}_k\|^2}\right), \quad (4.29)$$

then, by well known properties of twice continuously differentiable functions

with Lipschitz continuous gradients, we have that

$$\begin{aligned}
F(x_k + \xi^j d_k) &\leq F(x_k) + \xi^j \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k} + \frac{1}{2} \xi^{2j} \theta_{\max} \|[d_k]_{\mathcal{I}_k}\|^2 \\
&\leq F(x_k) + \xi^j \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k} + \xi^j (\eta - 1) \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k} \\
&= F(x_k) + \eta \xi^j \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k},
\end{aligned}$$

i.e., the inequality in line 13 will hold whenever (4.29) holds. On the other hand, suppose that $j > 0$ when line 7 in Algorithm 6 is reached. Then, since $k \in \mathcal{S}_\phi^{SD}$, we may conclude that

$$F(x_k + \alpha_B d_k) > F(x_k) + \eta \alpha_B \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k} = F(x_k) + \eta \alpha_B g_k^T \bar{d}_k \quad (4.30)$$

in line 10, because otherwise we would have $k \in \mathcal{S}_\phi^{ADD} = \mathcal{S}_\phi \setminus \mathcal{S}_\phi^{SD}$. Since no points of non-differentiability of $\|\cdot\|_1$ exist on the line segment connecting x_k to $x_k + \alpha_B d_k$ (which follows by the definition of α_B in line 8 of Algorithm 6), we can conclude for the same reason that we acquired (4.29) that (4.30) implies

$$\alpha_B > \frac{2(1 - \eta) |g_k^T \bar{d}_k|}{\theta_{\max} \|\bar{d}_k\|^2}.$$

Combining these two cases, we have that the line search procedure in Algorithm 6 will terminate with $x_{k+1} = x_k + \xi^j d_k$ where

$$\xi^j \geq \min \left\{ 1, \frac{2\xi(1 - \eta) |g_k^T \bar{d}_k|}{\theta_{\max} \|\bar{d}_k\|^2} \right\} \quad \text{and} \quad F(x_{k+1}) \leq F(x_k) + \eta \xi^j g_k^T \bar{d}_k. \quad (4.31)$$

Let us now consider two cases. First, suppose that $\xi^j = 1$ is returned from the line search, i.e., $j = 0$. Then, it follows from (4.31), lines 10 and 9 of Algo-

rithm 7, the Cauchy-Schwarz inequality, and Assumption 14 that

$$\begin{aligned} F(x_k) - F(x_{k+1}) &\geq -\eta \xi^j g_k^T \bar{d}_k = \eta |g_k^T \bar{d}_k| \geq \eta |g_k^T d_k^R| \\ &= \eta \alpha_k \|g_k\|^2 = \eta \frac{\|g_k\|^4}{g_k^T H_k g_k} \geq \frac{\eta}{\theta_{\max}} \|g_k\|^2. \end{aligned} \quad (4.32)$$

Now suppose that $\xi^j < 1$. Then, it follows from (4.31), the inequality $|g_k^T \bar{d}_k| \geq \|g_k\|^2 / \theta_{\max}$ established while deriving (4.32), and Lemma 23 that

$$\begin{aligned} F(x_k) - F(x_{k+1}) &\geq -\eta \xi^j g_k^T \bar{d}_k = \eta \xi^j |g_k^T \bar{d}_k| \geq \frac{2\eta \xi(1-\eta) |g_k^T \bar{d}_k|^2}{\theta_{\max} \|\bar{d}_k\|^2} \\ &\geq \frac{2\eta \xi(1-\eta) \theta_{\min}^2 \|g_k\|^4}{4\theta_{\max}^3 \|g_k\|^2} = \left(\frac{\eta \xi(1-\eta) \theta_{\min}^2}{2\theta_{\max}^3} \right) \|g_k\|^2. \end{aligned} \quad (4.33)$$

Combining (4.32) and (4.33) for the two cases establishes that

$$\begin{aligned} F(x_k) - F(x_{k+1}) &\geq \min \left\{ \frac{\eta}{\theta_{\max}}, \frac{\eta \xi(1-\eta) \theta_{\min}^2}{2\theta_{\max}^3} \right\} \|g_k\|^2 \\ &= \min \left\{ \frac{\eta}{\theta_{\max}}, \frac{\eta \xi(1-\eta) \theta_{\min}^2}{2\theta_{\max}^3} \right\} \|\nabla_{\mathcal{I}_k} F(x_k)\|^2 \\ &\geq \min \left\{ \frac{\eta}{\theta_{\max}}, \frac{\eta \xi(1-\eta) \theta_{\min}^2}{2\theta_{\max}^3} \right\} \|[\phi(x_k)]_{\mathcal{I}_k}\|^2 \\ &\geq \eta_\phi^2 \min \left\{ \frac{\eta}{\theta_{\max}}, \frac{\eta \xi(1-\eta) \theta_{\min}^2}{2\theta_{\max}^3} \right\} \|\phi(x_k)\|^2 \text{ for } k \in \mathcal{S}_\phi^{SD}, \end{aligned}$$

where we also used the condition in lines 7 of Algorithm 7 and the definition of $\phi(x_k)$. The inequality (4.28) follows from the previous inequality and $\|\beta(x_k)\| \leq \gamma \|\phi(x_k)\|$ for all $k \in \mathcal{S}_\phi \subseteq \mathcal{S}_\phi^{SD}$ as can be seen by line 6 of Algorithm 7. \square

We may now establish finiteness of the index set \mathcal{S}_ϕ^{SD} .

Lemma 25. *The index set \mathcal{S}_ϕ^{SD} is finite, i.e., $|\mathcal{S}_\phi^{SD}| < \infty$.*

Proof. To derive a contradiction, suppose that $|\mathcal{S}_\phi^{SD}| = \infty$, which means that Algorithm 7 does not terminate finitely. Thus, it follows from line 4 of Algorithm 7 that $\max\{\|\beta(x_k)\|, \|\phi(x_k)\|\} > \epsilon$ for all $k \geq 0$. Also, it follows from Lemmas 21

and [22](#) that there exists an iteration number k_1 such that $k \in \mathcal{S}_\phi^{SD}$ for all $k \geq k_1$. Thus, with Lemma [24](#), we have for all $\ell \geq k_1$ that

$$\begin{aligned}
F(x_{k_1}) - F(x_{\ell+1}) &= \sum_{k=k_1}^{\ell} [F(x_k) - F(x_{k+1})] \\
&= \sum_{k \in \mathcal{S}_\phi^{SD}, k_1 \leq k \leq \ell} [F(x_k) - F(x_{k+1})] \\
&\geq \sum_{k \in \mathcal{S}_\phi^{SD}, k_1 \leq k \leq \ell} \kappa_\phi \max\{\gamma^{-2} \|\beta(x_k)\|^2, \|\phi(x_k)\|^2\} \\
&\geq \sum_{k \in \mathcal{S}_\phi^{SD}, k_1 \leq k \leq \ell} \kappa_\phi \min\{\gamma^{-2}, 1\} \epsilon^2.
\end{aligned}$$

Rearranging the previous inequality shows that

$$\begin{aligned}
\lim_{\ell \rightarrow \infty} F(x_{\ell+1}) &\leq \lim_{\ell \rightarrow \infty} \left[F(x_{k_1}) - \sum_{k \in \mathcal{S}_\phi^{SD}, k_1 \leq k \leq \ell} \kappa_\phi \min\{\gamma^{-2}, 1\} \epsilon^2 \right] \\
&= F(x_{k_1}) - \sum_{k \in \mathcal{S}_\phi^{SD}, k_1 \leq k} \kappa_\phi \min\{\gamma^{-2}, 1\} \epsilon^2 = -\infty,
\end{aligned}$$

which contradicts Assumption [14](#). Thus, we conclude that $|\mathcal{S}_\phi^{SD}| < \infty$. \square

We now prove our first main convergence result.

Theorem 26. *Algorithm [7](#) terminates finitely.*

Proof. Since each iteration number k generated in the algorithm is an element of $\mathcal{S}_\beta \cup \mathcal{S}_\phi^{ADD} \cup \mathcal{S}_\phi^{SD}$, the result follows by Lemmas [21](#), [22](#), and [25](#). \square

Our final convergence result states what happens when the finite termination criterion is removed from Algorithm [7](#).

Theorem 27. *Let x_* be the unique solution to problem [\(1.1\)](#). If ϵ in the finite termination condition in line [4](#) of Algorithm [7](#) is replaced by zero, then either:*

(i) *there exists an iteration k such that $x_k = x_*$; or*

(ii) *infinitely many iterations $\{x_k\}$ are computed and they satisfy*

$$\lim_{k \rightarrow \infty} x_k = x_*, \quad \lim_{k \rightarrow \infty} \varphi(x_k) = 0, \quad \text{and} \quad \lim_{k \rightarrow \infty} \beta(x_k) = 0.$$

Proof. If case (i) occurs, then there is nothing left to prove. Thus, for the remainder of the proof, we assume that case (i) does not occur. Since case (i) does not occur, we know that Algorithm 7 performs an infinite sequence of iterations. Let us then define the set $\mathcal{S} := \mathcal{S}_\beta \cup \mathcal{S}_\phi^{SD}$, which must be infinite (since any consecutive sequence of iterations in \mathcal{S}_ϕ^{ADD} must be finite by the finiteness of n). It follows from (4.24) for $k \in \mathcal{S}_\beta$, (4.28) for $k \in \mathcal{S}_\phi^{SD}$, and Assumption 14 (specifically, the assumption that f is bounded below on \mathcal{L}) that

$$\lim_{k \in \mathcal{S}} \max\{\|\beta(x_k)\|, \|\varphi(x_k)\|\} = 0.$$

Combining this with Assumption 14 and Lemma 13 gives

$$\lim_{k \in \mathcal{S}} x_k = x_*. \tag{4.34}$$

We claim that the previous limit holds over all iterations. To prove this by contradiction, suppose there exists an infinite $\mathcal{K} \subseteq \mathcal{S}_\phi^{ADD}$ and scalar $\varepsilon > 0$ with

$$\|x_k - x_*\| \geq \varepsilon \quad \text{for all } k \in \mathcal{K}. \tag{4.35}$$

From Assumption 14, we conclude that there exists $\delta > 0$ such that

$$\text{if } F(x) \leq F(x_*) + \delta, \text{ then } \|x - x_*\| < \varepsilon. \tag{4.36}$$

Also, from (4.34) and Assumption 14, there exists a smallest $k_S \in \mathcal{S}$ so that

$$F(x_{k_S}) \leq F(x_*) + \delta. \quad (4.37)$$

There then exists a smallest $k_K \in \mathcal{K}$ such that $k_K > k_S$. Since, by construction, $\{F(x_k)\}_{k \geq 0}$ is monotonically decreasing, we may conclude with (4.37) that

$$F(x_{k_K}) \leq F(x_{k_S}) \leq F(x_*) + \delta. \quad (4.38)$$

Combining (4.38) and (4.36), we deduce that $\|x_{k_K} - x_*\| < \epsilon$, which contradicts (4.35) since $k_K \in \mathcal{K}$. This completes the proof. \square

4.2.2 Local convergence analysis

In this section, we prove a super-linear local convergence rate guarantee for FaRSA. For the analysis, we assume in this section that case (i) of Theorem 27 does not occur and that $\epsilon = 0$ in Algorithm 7. Together, these assumptions imply that an infinite sequence of iterates $\{x_k\}_{k \geq 0}$ is computed by Algorithm 7.

A feature of FaRSA that we will show is that the iterates reveal the variables that are zero, negative, and positive at the solution x_* after a finite number of iterations. Establishing this fact requires the following constants:

$$0 < \delta_1 := \frac{1}{2} \min_{i \in \mathcal{I}^\pm(x_*)} |[x_*]_i|$$

and $0 < \delta_2 := \frac{1}{2} \min_{i \in \mathcal{I}^0(x_*)} (\lambda - |\nabla_i f(x_*)|).$

We remark that δ_2 is positive as a consequence of Assumption 17.

The next result shows that the activities at x_k agree with those at x_* for certain components when x_k is sufficiently close to x_* . (The word *activities* refers to the partition of a vector into its positive, negative, and zero variables.)

Lemma 28. *If $\|x_k - x_*\| \leq \delta_1/2$, then*

$$\text{sgn}([x_k]_i) = \text{sgn}([x_*]_i) \text{ for all } i \in (\mathcal{I}^0(x_*) \cap \mathcal{I}^0(x_k)) \cup \mathcal{I}^\pm(x_*).$$

Proof. The conclusion holds for $i \in \mathcal{I}^0(x_*) \cap \mathcal{I}^0(x_k)$ from the definition of \mathcal{I}^0 . To obtain the conclusion for $i \in \mathcal{I}^\pm(x_*)$, observe that since $\|x_k - x_*\| \leq \delta_1/2$ by assumption, it follows from the definition of δ_1 , the triangle inequality, and basic norm inequalities that

$$\begin{aligned} \text{for all } i \in \mathcal{I}^-(x_*): [x_k]_i &= [x_k]_i - [x_*]_i + [x_*]_i \leq |[x_k - x_*]_i| - 2\delta_1 \\ &\leq \|x_k - x_*\| - 2\delta_1 \leq \delta_1/2 - 2\delta_1 = -(3/2)\delta_1. \end{aligned} \quad (4.39)$$

Similarly,

$$\begin{aligned} \text{for all } i \in \mathcal{I}^+(x_*): [x_k]_i &= [x_k]_i - [x_*]_i + [x_*]_i \geq -|[x_k - x_*]_i| + 2\delta_1 \\ &\geq -\|x_k - x_*\| + 2\delta_1 \geq -\delta_1/2 + 2\delta_1 = (3/2)\delta_1. \end{aligned} \quad (4.40)$$

The conclusion holds for $i \in \mathcal{I}^\pm(x_*)$ because of (4.39) and (4.40). \square

We now bound the magnitudes of the components of the gradient of f for elements in $\mathcal{I}^0(x_*)$ and prove that $\beta(x_k) = 0$ when x_k is sufficiently close to x_* .

Lemma 29. *If $\|x_k - x_*\| \leq \min\{\delta_1/2, \delta_2/L_g\}$, then the following hold:*

- (i) $|\nabla_i f(x_k)| < \lambda - \delta_2$ for all $i \in \mathcal{I}^0(x_*)$,
- (ii) $\beta(x_k) = 0$, and
- (iii) $k \in \mathcal{S}_\phi$.

Proof. It follows from the triangle inequality, basic norm inequalities, Lipschitz

continuity of ∇f , and the assumption of the lemma that

$$\begin{aligned}
|\nabla_i f(x_k)| - |\nabla_i f(x_*)| &\leq |\nabla_i f(x_k) - \nabla_i f(x_*)| \\
&\leq \|\nabla f(x_k) - \nabla f(x_*)\| \\
&\leq L_g \|x_k - x_*\| \leq L_g(\delta_2/L_g) = \delta_2 \quad \text{for all } i \in \{1, \dots, n\}.
\end{aligned}$$

It follows from this string of inequalities that

$$|\nabla_i f(x_k)| \leq |\nabla_i f(x_*)| + \delta_2 \quad \text{for all } i \in \{1, \dots, n\}. \quad (4.41)$$

Defining $c := \max_{i \in \mathcal{I}^0(x_*)} |\nabla_i f(x_*)|$, the definition of δ_2 implies that

$$\delta_2 = \frac{\lambda - c}{2}.$$

Combining this with (4.41) and the definition of c shows that

$$|\nabla_i f(x_k)| \leq |\nabla_i f(x_*)| + \delta_2 \leq c + \frac{\lambda - c}{2} = \frac{\lambda + c}{2} = \lambda - \delta_2 \quad \text{for all } i \in \mathcal{I}^0(x_*),$$

which proves part (i). Now, to prove part (ii), observe that we can only have $[\beta(x_k)]_i \neq 0$ if $i \in \mathcal{I}^0(x_k)$, where $\mathcal{I}^0(x_k) = (\mathcal{I}^0(x_*) \cap \mathcal{I}^0(x_k)) \cup (\mathcal{I}^\pm(x_*) \cap \mathcal{I}^0(x_k))$. However, from Lemma 28 and the assumption of the lemma, we have that $\mathcal{I}^\pm(x_*) \cap \mathcal{I}^0(x_k) = \emptyset$, while for $i \in \mathcal{I}^0(x_*) \cap \mathcal{I}^0(x_k)$ we have from part (i) that $[\beta(x_k)]_i = 0$. Thus, overall, $\beta(x_k) = 0$, as desired. Finally, part (iii) follows from part (ii) and line 6 of Algorithm 7. \square

Using the previous result allows us to prove the following lemma, which establishes that the activities of the iterates eventually do not change.

Lemma 30. *There exists an iteration \bar{k} so that the following hold for all $k \geq \bar{k}$:*

$$(i) \quad \|x_k - x_*\| \leq \min\{\delta_1/2, \delta_2/L_g\},$$

(ii) $k \in \mathcal{S}_\phi^{SD}$, and

(iii) $\mathcal{I}^0(x_k) = \mathcal{I}^0(x_{\bar{k}})$, $\mathcal{I}^+(x_k) = \mathcal{I}^+(x_{\bar{k}})$, and $\mathcal{I}^-(x_k) = \mathcal{I}^-(x_{\bar{k}})$.

Proof. First, the fact that (i) holds for all sufficiently large k follows from Theorem 27(ii). Second, to prove that (ii) holds for sufficiently large k , observe that (i) and Lemma 29 imply that $k \in \mathcal{S}_\phi$ for all sufficiently large k . The fact that there exists an iteration \bar{k} such that $k \in \mathcal{S}_\phi^{SD}$ for all $k \geq \bar{k}$ now follows because (a) $\mathcal{S}_\phi = \mathcal{S}_\phi^{SD} \cup \mathcal{S}_\phi^{ADD}$, (b) for each $k \in \mathcal{S}_\phi^{ADD}$ at least one nonzero component of x_k becomes zero at x_{k+1} and no components that were zero at x_k become nonzero at x_{k+1} , and (c) for each $k \in \mathcal{S}_\phi^{SD}$ the activities at x_k are the same as the activities at x_{k+1} . Finally, using (ii) and the fact that the activities at x_k are the same as the activities at x_{k+1} for $k \in \mathcal{S}_\phi^{SD}$ shows that (iii) holds. \square

Our next aim is to establish that the activities at x_k for $k \geq \bar{k}$ are the same as the activities at the solution x_* . We require the following lemma.

Lemma 31. *For \bar{k} defined in Lemma 30, it follows that*

$$\lim_{\bar{k} \leq k \rightarrow \infty} \|g_k\| = 0 \text{ and } \lim_{\bar{k} \leq k \rightarrow \infty} \|\bar{d}_k\| = 0.$$

Proof. We first observe from Lemma 30(ii) that $k \in \mathcal{S}_\phi^{SD}$ for all $k \geq \bar{k}$. Thus, it follows as written in the proof of Lemma 24 that

$$F(x_{k+1}) \leq F(x_k) - \min \left\{ \frac{\eta}{\theta_{\max}}, \frac{\eta\xi(1-\eta)\theta_{\min}^2}{2\theta_{\max}^3} \right\} \|g_k\|^2 \text{ for all } k \geq \bar{k}.$$

This inequality and the fact that F is bounded below as a consequence of Assumption 14 allows us to deduce that $\lim_{k \rightarrow \infty} \|g_k\| = 0$, which is the first desired limit. Combining the first limit with Lemma 23 and $\mathcal{S}_\phi^{SD} \subseteq \mathcal{S}_\phi$ allows us to conclude that $\lim_{k \rightarrow \infty} \|\bar{d}_k\| = 0$, which completes the proof. \square

We now prove a finite active set identification result. Since we already know from Lemma 30(iii) that the activities of the sequence $\{x_k\}$ do not change for $k \geq \bar{k}$, all that remains is to show that they agree with the activities of x_* .

Lemma 32. *If we choose $\mathcal{I}_k = \{i \in \mathcal{I} : [\phi(x_k)]_i \neq 0\}$ in line 7 of Algorithm 7 for all $k \geq \bar{k}$ with \bar{k} defined in Lemma 30, then it holds that*

$$\mathcal{I}^0(x_k) = \mathcal{I}^0(x_*), \quad \mathcal{I}^+(x_k) = \mathcal{I}^+(x_*), \quad \text{and} \quad \mathcal{I}^-(x_k) = \mathcal{I}^-(x_*) \quad \text{for all } k \geq \bar{k}.$$

Proof. We observe from Lemma 31 and the choice for g_k in Algorithm 7 that

$$0 = \lim_{\bar{k} \leq k \rightarrow \infty} \|\nabla_{\mathcal{I}_k} F(x_k)\| = \lim_{\bar{k} \leq k \rightarrow \infty} \|\nabla f(x_k) + \lambda \zeta\|_{\mathcal{I}_k}, \quad (4.42)$$

where

$$[\zeta]_i := \begin{cases} 1 & \text{if } i \in \mathcal{I}^+(x_{\bar{k}}), \\ -1 & \text{if } i \in \mathcal{I}^-(x_{\bar{k}}), \\ \text{arbitrary} & \text{if } i \in \mathcal{I}^0(x_{\bar{k}}). \end{cases}$$

(We remind the reader that from the choice of \mathcal{I}_k in the statement of this lemma, the definition of ϕ , and Lemma 30(iii) that $\mathcal{I}_k = \{i \in \mathcal{I} : [\phi(x_k)]_i \neq 0\} \subseteq \mathcal{I}^\pm(x_k) = \mathcal{I}^\pm(x_{\bar{k}})$ for all $k \geq \bar{k}$. In particular, this means that $\nabla_{\mathcal{I}_k} F(x_k)$ is well-defined in (4.42).)

Next, as an intermediate result, we claim that $\mathcal{I}_k \cap \mathcal{I}^0(x_*) = \emptyset$ for all sufficiently large k . For a proof by contradiction, suppose that there exists an infinite subsequence of iteration numbers \mathcal{K} and an index j such that $j \in \mathcal{I}_k \cap \mathcal{I}^0(x_*)$ for all $k \in \mathcal{K}$. It follows from $j \in \mathcal{I}^0(x_*)$ that $[x_*]_j = 0$. On the other hand, from $j \in \mathcal{I}_k$ for all $k \in \mathcal{K}$ and (4.42), it follows that $0 = \lim_{\bar{k} \leq k \in \mathcal{K}} [\nabla f(x_k) + \lambda \zeta]_j = [\nabla f(x_*) + \lambda \zeta]_j$, which implies that $|\nabla_j f(x_*)| = \lambda$. This violates Assumption 17 since $j \in \mathcal{I}^0(x_*)$, which means that we must conclude that $\mathcal{I}_k \cap \mathcal{I}^0(x_*) = \emptyset$ for all sufficiently large k , as claimed.

Next, to prove that $\mathcal{I}^0(x_*) \subseteq \mathcal{I}^0(x_k)$ for all $k \geq \bar{k}$, let $j \in \mathcal{I}^0(x_*)$. Since we have shown that $\mathcal{I}_k \cap \mathcal{I}^0(x_*) = \emptyset$ for all sufficiently large k , we can conclude that $j \notin \mathcal{I}_k$ for all sufficiently large k . This may be combined with $k \in \mathcal{S}_\phi^{SD}$ for all $k \geq \bar{k}$ (recall Lemma 30(ii)) and line 11 of Algorithm 7 to conclude that $[d_k]_j = 0$ for all sufficiently large k . Since the update procedure for Algorithm 7 is given by $x_{k+1} = x_k + \alpha_k d_k$ for some positive α_k , we can conclude that $[x_{k+1}]_j = [x_k]_j$ for all sufficiently large k . However, since we also know from Theorem 27(ii) that $\{x_k\} \rightarrow x_*$, we can see that $[x_k]_j = [x_*]_j = 0$ for all sufficiently large k . Finally, if we observe that the activities of $\{x_k\}$ do not change for $k \geq \bar{k}$ (see Lemma 30(iii)), we can conclude the stronger statement that $[x_k]_j = 0$ for all $k \geq \bar{k}$, i.e., that $j \in \mathcal{I}^0(x_k)$ for all $k \geq \bar{k}$. Overall, we have established that $\mathcal{I}^0(x_*) \subseteq \mathcal{I}^0(x_k)$ for all $k \geq \bar{k}$.

To prove the opposite inclusion, i.e., that $\mathcal{I}^0(x_k) \subseteq \mathcal{I}^0(x_*)$ for all $k \geq \bar{k}$, suppose that $j \in \mathcal{I}^0(x_{\hat{k}})$ for some $\hat{k} \geq \bar{k}$. It follows from Lemma 30(iii) that $j \in \mathcal{I}^0(x_k)$ for all $k \geq \bar{k}$, meaning that $[x_k]_j = 0$ for all $k \geq \bar{k}$. Then, if we use the fact that $\{x_k\} \rightarrow x_*$ (from Theorem 27(ii)), we must conclude that $[x_*]_j = 0$, i.e., that $j \in \mathcal{I}^0(x_*)$. Thus, we have shown that $\mathcal{I}^0(x_k) \subseteq \mathcal{I}^0(x_*)$ for all $k \geq \bar{k}$. Combining this with the conclusion from the previous paragraph yields $\mathcal{I}^0(x_k) = \mathcal{I}^0(x_*)$ for all $k \geq \bar{k}$, which is the first desired result.

We next prove that $\mathcal{I}^+(x_k) = \mathcal{I}^+(x_*)$ for all $k \geq \bar{k}$. The inclusion $\mathcal{I}^+(x_*) \subseteq \mathcal{I}^+(x_k)$ for all $k \geq \bar{k}$ follows from Lemma 28 and Lemma 30(i). Thus, it remains to prove the opposite inclusion that $\mathcal{I}^+(x_k) \subseteq \mathcal{I}^+(x_*)$ for all $k \geq \bar{k}$. For this purpose, suppose that $j \in \mathcal{I}^+(x_{\hat{k}})$ for some $\hat{k} \geq \bar{k}$, which because of Lemma 30(iii) implies that $j \in \mathcal{I}^+(x_k)$ for all $k \geq \bar{k}$, i.e., that $[x_k]_j > 0$ for all $k \geq \bar{k}$. Since $\{x_k\} \rightarrow x_*$ (from Theorem 27), this means that $[x_*]_j \geq 0$ so that

$$j \in \mathcal{I}^0(x_*) \cup \mathcal{I}^+(x_*) = \mathcal{I}^0(x_k) \cup \mathcal{I}^+(x_*) \text{ for all } k \geq \bar{k}; \quad (4.43)$$

here, we have used the already-proved fact that $\mathcal{I}^0(x_k) = \mathcal{I}^0(x_*)$ for all $k \geq \bar{k}$. Since from above we know that $j \in \mathcal{I}^+(x_k)$ for all $k \geq \bar{k}$, and that by construction $\mathcal{I}^+(x_k) \cap \mathcal{I}^0(x_k) = \emptyset$, we can conclude from (4.43) that $j \in \mathcal{I}^+(x_*)$. Thus, we have shown that $\mathcal{I}^+(x_k) \subseteq \mathcal{I}^+(x_*)$ for all $k \geq \bar{k}$. Since we have established both inclusions, we have proved that $\mathcal{I}^+(x_k) = \mathcal{I}^+(x_*)$ for all $k \geq \bar{k}$.

The final result, namely that $\mathcal{I}^-(x_k) = \mathcal{I}^-(x_*)$ for all $k \geq \bar{k}$, may be proved using the same approach as in the previous paragraph. \square

Before giving a local rate of convergence result for Algorithm 7, we must establish that the unit stepsize is accepted for sufficiently large k .

Lemma 33. *The iterate update computed by Algorithm 7 is given by*

$$x_{k+1} = x_k + d_k \text{ for all sufficiently large } k,$$

where the vector d_k is defined in line 11 using the vector \bar{d}_k computed to satisfy the conditions in line 10.

Proof. Lemma 30(ii), Lemma 31, and line 11 of Algorithm 7 give $\lim_{k \rightarrow \infty} \|d_k\| = 0$. Combining this with Lemma 32 and $\{x_k\} \rightarrow x_*$ (see Theorem 27) shows that $x_k + d_k$ is in the same orthant as x_k for all sufficiently large k , i.e. that $\text{sgn}(x_k + d_k) = \text{sgn}(x_k)$ for all sufficiently large k . Since we know that $k \in \mathcal{S}_\phi^{SD}$ for all sufficiently large k (see Lemma 30(ii)), we know that Algorithm 7 calls the line search Algorithm 6 for all sufficiently large k . Moreover, since $\text{sgn}(x_k + d_k) = \text{sgn}(x_k)$ for all sufficiently large k , Algorithm 6 will find that $j = 0$ when line 7 is reached, which means that the condition in line 13 will first be checked. In what follows, we proceed to show that this condition in line 13 will indeed be satisfied with $j = 0$, thereby showing that $x_{k+1} = x_k + d_k$ and completing the proof of the lemma.

To prove that the condition in line 13 of Algorithm 6 is satisfied with $j = 0$, we first notice from lines 9 and 10 of Algorithm 7, Assumption 14, the definition of H_k , and the interlacing property of the eigenvalues of submatrices that

$$|g_k^T \bar{d}_k| \geq |g_k^T d_k^R| = \frac{\|g_k\|^4}{g_k^T H_k g_k} \geq \frac{\|g_k\|^4}{\|H_k\| \|g_k\|^2} \geq \frac{\|g_k\|^2}{\theta_{\max}}. \quad (4.44)$$

Next, from the Mean Value Theorem, the fact that F is twice continuously differentiable on the interval $[x_k, x_k + d_k]$, the definitions of g_k and \bar{d}_k , and the fact that the second derivatives of the regularizer are zero for $i \in \mathcal{I}_k$, there exists $z_k \in [x_k, x_k + d_k]$ such that

$$\begin{aligned} F(x_k + d_k) &= F(x_k) + \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k} + \frac{1}{2} [d_k]_{\mathcal{I}_k}^T \nabla_{\mathcal{I}_k \mathcal{I}_k}^2 f(z_k) [d_k]_{\mathcal{I}_k} \\ &= F(x_k) + g_k^T \bar{d}_k + \frac{1}{2} \bar{d}_k^T \nabla_{\mathcal{I}_k \mathcal{I}_k}^2 f(z_k) \bar{d}_k. \end{aligned}$$

Combining this with Assumption 16, Theorem 27(ii), $\lim_{k \rightarrow \infty} \|d_k\| = 0$, line 10 in Algorithm 7, and Lemma 23 gives

$$\begin{aligned} F(x_k + d_k) - F(x_k) - \frac{1}{2} g_k^T \bar{d}_k &= \frac{1}{2} (g_k^T \bar{d}_k + \bar{d}_k^T \nabla_{\mathcal{I}_k \mathcal{I}_k}^2 f(z_k) \bar{d}_k) \\ &= \frac{1}{2} (g_k^T \bar{d}_k + \bar{d}_k^T H_k \bar{d}_k) + \frac{1}{2} (\bar{d}_k^T (\nabla_{\mathcal{I}_k \mathcal{I}_k}^2 f(z_k) - H_k) \bar{d}_k) \\ &\leq \frac{1}{2} \bar{d}_k^T (H_k \bar{d}_k + g_k) + \frac{1}{2} L_H \|\bar{d}_k\|^3 \\ &\leq \frac{1}{2} \|\bar{d}_k\| \|H_k \bar{d}_k + g_k\| + \frac{1}{2} L_H \|\bar{d}_k\|^3 \\ &\leq \frac{1}{2} \|\bar{d}_k\| \|g_k\|^2 + \frac{2L_H}{\theta_{\min}^2} \|\bar{d}_k\| \|g_k\|^2 \\ &= \left(\frac{1}{2} + \frac{2L_H}{\theta_{\min}^2}\right) \|\bar{d}_k\| \|g_k\|^2 \text{ for all sufficiently large } k. \end{aligned} \quad (4.45)$$

Since $\eta \in (0, \frac{1}{2})$, we know from Lemma 31 and Assumption 14 that

$$\theta_{\max} \left(\frac{1}{2} + \frac{2L_H}{\theta_{\min}^2}\right) \|\bar{d}_k\| \leq \frac{1}{2} (1 - 2\eta) \text{ for all sufficiently large } k. \quad (4.46)$$

It now follows from (4.45), (4.44), (4.46), and $g_k^T \bar{d}_k < 0$ that

$$\begin{aligned}
F(x_k + d_k) - F(x_k) &\leq \frac{1}{2} g_k^T \bar{d}_k + \left(\frac{1}{2} + \frac{2L_H}{\theta_{\min}^2}\right) \|\bar{d}_k\| \|g_k\|^2 \\
&\leq \frac{1}{2} g_k^T \bar{d}_k + \theta_{\max} \left(\frac{1}{2} + \frac{2L_H}{\theta_{\min}^2}\right) \|\bar{d}_k\| |g_k^T \bar{d}_k| \\
&\leq \frac{1}{2} g_k^T \bar{d}_k + \frac{1}{2} (1 - 2\eta) |g_k^T \bar{d}_k| \\
&= \frac{1}{2} (1 - (1 - 2\eta)) g_k^T \bar{d}_k \\
&= \eta g_k^T \bar{d}_k \text{ for all sufficiently large } k, \tag{4.47}
\end{aligned}$$

which after rearrangement and use of the definitions of g_k and \bar{d}_k implies that

$$F(x_k + d_k) \leq F(x_k) + \eta \nabla_{\mathcal{I}_k} F(x_k)^T [d_k]_{\mathcal{I}_k}. \tag{4.48}$$

Therefore, we have shown that the condition in line 13 is satisfied when $j = 0$ for all sufficiently large k , thereby completing the proof of this lemma. \square

We showed in Lemma 32 that, after a finite number of iterations, the signs of the variables at x_k and x_* are the same. Moreover, Lemma 33 shows that for sufficiently large k , the iteration for Algorithm 7 takes the form $x_{k+1} = x_k + d_k$, where d_k is defined in line 11 using \bar{d}_k computed to satisfy the conditions in line 10. Therefore, the iterations of Algorithm 7 in the space of nonzero variables are exactly those of an inexact Newton method for computing a zero of ∇f restricted to the components in the set $\mathcal{I}^\pm(x_*)$. Consequently, the next result follows from [67, Theorem 3.3].

Lemma 34. *Let us define*

$$\bar{x}_k := [x_k]_{\mathcal{I}^\pm(x_*)} \text{ and } \bar{x}_* := [x_*]_{\mathcal{I}^\pm(x_*)}.$$

Suppose that we choose $\mathcal{I}_k = \{i \in \mathcal{I} : [\phi(x_k)]_i \neq 0\}$ in line 7 of Algorithm 7 for

all sufficiently large k . Then, if either $r \in (1, 2]$, or $r = 1$ and $\{\mu_k\} \rightarrow 0$, then $\{\bar{x}_k\} \rightarrow \bar{x}_*$ at a superlinear rate. In particular, if we choose $r = 2$, then the rate is quadratic.

We now state our final rate-of-convergence result.

Theorem 35. Suppose that $\mathcal{I}_k = \{i \in \mathcal{I} : [\phi(x_k)]_i \neq 0\}$ in line 7 of Algorithm 7 for all sufficiently large k . Then, if either $r \in (1, 2]$, or $r = 1$ and $\{\mu_k\} \rightarrow 0$, then $\{x_k\} \rightarrow x_*$ at a superlinear rate. In particular, if we choose $r = 2$, then the rate is quadratic.

Proof. The result follows from Lemma 34 and since $[x_k]_{\mathcal{I}^0(x_*)} = [x_*]_{\mathcal{I}^0(x_*)} = 0$ for all $k \geq \bar{k}$ (with \bar{k} defined in Lemma 30) as a result of Lemma 32. \square

Chapter 5

Numerical Experiments

In Chapter 4, we present our proposed algorithm FaRSA, with theoretical support in terms of global convergence and local convergence properties. It is now natural to ask whether FaRSA performs better than other popular solvers for ℓ_1 -regularization problems. For answering this question, in this section we compare FaRSA with other popular solvers by testing them on numerous data sets to numerically demonstrate the effectiveness of FaRSA. The comparison will involve three aspects. First, we concentrate on comparing two common evaluation metrics for optimization algorithms, namely the run time and final objective function value obtained. Second, the local convergent performance of the solvers is evaluated. Third, we investigate how well solutions produced by FaRSA on training data generalize to unseen test data for binary classification.

We structure this chapter as followings. In Section 5.1, we give implementation details for the FaRSA software, including default parameter settings, acceleration strategy, etc. Detailed information concerning the solvers tested for comparison purposes are provided in Section 5.2. The experimental settings are provided in Section 5.3, with the data sets used given in Section 5.4. Finally, the results of our numerical experiments are presented in Section 5.5.

5.1 FaRSA Software

We have developed C and Matlab implementations of FaRSA that are freely available. We summarize the main features of the software in this section. In Section 5.1.1 we present various implementation details such as default parameter settings, specifics on how to implement the various steps in Algorithm 7, and auxiliary algorithms implemented in FaRSA. The computational details for a couple of popular objective functions are discussed in Section 5.1.2.

5.1.1 Implementation details

The default parameter values for FaRSA are chosen based on a rough tuning procedure that resulted in the following. We use $\gamma = 1$ in Step 6 of Algorithm 7 so that no preference is given to whether the k th iteration is in \mathcal{S}_β or \mathcal{S}_ϕ . In Step 7, we use $\mathcal{I}_k \equiv \{i : [\phi(x_k)]_i \neq 0\}$, which is equivalent to choosing $\eta_\phi = 1$. However, our choice for \mathcal{I}_k in Step 14 is based on taking the largest (in absolute value) 80% (rounded up to the nearest integer) of the entries from β , which means that $\|[\beta(x_k)]_{\mathcal{I}_k}\| \geq \|\beta(x_k)\|_\infty \geq (1/\sqrt{n})\|\beta(x_k)\|_2$ so that the required condition in Step 14 holds with $\eta_\beta = 1/\sqrt{n}$. For efficiently computing the largest 80% entries, we implemented a three-way partitioning variant of the quick sort algorithm to address the possibility of duplicate entries [68], and set the median element as the starting pivot in order to improve the practical performance of sorting [69, Chapter 2].

For the line search procedures stated as Algorithm 6 and Algorithm 4, which are called in Steps 12 and 16 of Algorithm 7, we use $\eta = 10^{-2}$ and $\xi = 0.5$. As an initial solution estimate we use $x_0 = 0$, and we terminate if x_k satisfies

$$\max\{\|\beta(x_k)\|, \|\phi(x_k)\|\} \leq \epsilon \max\{1, \|\beta(x_0)\|, \|\phi(x_0)\|\}$$

with $\epsilon = 10^{-6}$, which is based on Step 4 of Algorithm 7.

Our strategy for computing the search direction d_k when $k \in \mathcal{S}_\phi$ is the one that we found to yield the best practical performance. As is well known, the CG algorithm is a popular and effective choice for computing an approximate solution to the linear system $H_k \bar{d} = -g_k$ associated with the data in Step 8 of Algorithm 7. Our strategy is based on monitoring the CG iterates and terminating when certain conditions are satisfied. In order to describe these conditions for the k th outer iteration, let $d_{j,k}$ denote the j th CG iterate, $r_{j,k} = \|H_k d_{j,k} + g_k\|$ denote the j th CG residual, and $v_{j,k}$ denote the number of entries in the vector $[x_k]_{\mathcal{I}_k} + d_{j,k}$ that fall into a different orthant than $[x_k]_{\mathcal{I}_k}$. We decide to terminate the CG method when any of the following hold:

$$v_{j,k} \geq \min\{10^7, 0.25|\mathcal{I}_k|\}; \text{ or} \quad (5.1a)$$

$$\|d_{j,k}\| \geq \delta_{k,\phi} := \max\{10^{-3}, \min\{10^3, 10\|x_{k_\phi(k)+1} - x_{k_\phi(k)}\|\}\}; \text{ or} \quad (5.1b)$$

$$r_{j,k} \leq \max\{10^{-1} \min\{r_{0,k}, r_{0,k}^2\}, 10^{-12}\}; \text{ or} \quad (5.1c)$$

$$r_{j,k} \leq \max\{10^{-1} r_{0,k}, 10^{-12}\} \text{ and either } v_{j,k-1} > 0 \text{ or } \|\beta(x_{k-1})\| > \epsilon; \quad (5.1d)$$

where $k_\phi(k) := \max\{\bar{k} : \bar{k} \in \mathcal{S}_\phi \text{ and } \bar{k} < k\}$ and $\epsilon = 10^{-6}$. When the inequality in (5.1a) holds, it indicates that a relatively large number of entries in $[x_k]_{\mathcal{I}_k} + d_{j,k}$ are in a different orthant than $[x_k]_{\mathcal{I}_k}$. In this case, it makes sense to terminate CG since the exact solution is likely to be in a different orthant. In the second termination condition (5.1b), the value $\delta_{k,\phi}$ plays a role similar to a trust region constraint for trust region methods; in particular, the value $\|x_{k_\phi(k)+1} - x_{k_\phi(k)}\|$ is the size of the step computed during the last iteration in \mathcal{S}_ϕ . When the inequality in (5.1b) tests true, the size of the CG trial iterate $d_{j,k}$ is deemed to be relatively large. In this case it makes sense to terminate CG because it is unlikely that x_k is “near” a minimizer to the optimization problem.

(We remind the reader of the property that, if initialized with the zero vector, the iterates produced by CG are monotonically increasing in terms of the Euclidean norm.) In some sense, the best termination outcome is (5.1c), which requires (ignoring the numerical safeguard value of 10^{-12}) that the residual of the Newton linear system is sufficiently reduced. In particular, condition (5.1c) aims to achieve a quadratic decrease in the residual (confer with (4.19)) as a minimizer of the optimization problem is approached, which allows for the superlinear rate of convergence that we established in Theorem 35. Finally, we also allow for termination if the conditions in (5.1d) hold. These conditions include a relaxed requirement on the reduction of the CG residual as compared to (5.1c), which we allow since when either $v_{j,k-1} > 0$ or $\|\beta(x_{k-1})\| > \epsilon$, we expect that x_k is not near a minimizer. For context, note that Lemma 29 shows that $\beta(x_k)$ would be zero when x_k is close enough to a minimizer x_* , and Lemma 32 shows that all iterates corresponding to sufficiently large iteration numbers would lie in the same orthant as x_* . Overall, the relaxed requirement on the CG residual in (5.1d) aims to avoid excessive computation.

5.1.2 Objective functions supported

Currently, FaRSA supports generic objective functions f to allow users to input personalized objective functions, as well as two additional routines for two popular objective functions, namely the logistic loss and least-squares loss. These three options are discussed in more detail over the next three sections.

5.1.2.1 Generic objective function

Our software allows users to use any convex objective function that they desire, i.e., a generic objective function. To use such a generic function, the user must supply three function interfaces associated with their personal objective

function. Specifically, they must supply the following: (i) a function that computes the objective function value for a given x ; (ii) a function that computes the gradient of f at given x ; and (iii) a function that computes the product of the Hessian of f at a given point x with a desired vector, which in our implementation is used by our CG-based subproblem solver.

5.1.2.2 Logistic loss

Perhaps the most popular loss function used in data analysis is the logistic function. For this reason, we have included it as a built-in function so that the user does not need to supply any objective function related subroutines such as those in Section 5.1.2.1 for the generic objective function case.

The logistic objective function is defined in (1.6). Here, let us define a version of the logistic function that does not explicitly include the “bias” term, i.e.,

$$f(x) := \frac{1}{N} \sum_{\ell=1}^N \log(1 + e^{-y_{\ell} d_{\ell}^T x}) \quad (5.2)$$

where $d_{\ell} \in \mathbb{R}^n$ is the i th data vector, d_{ℓ}^T is ℓ th row in the data matrix D , N is the number of data vectors in the dataset, and $y_{\ell} \in \{-1, 1\}$ is the class label for the ℓ th data vector. In order to obtain the derivatives of f as defined in (5.2), let us define $\tau(\zeta) := 1/(1 + e^{-\zeta})$. We can now observe that

$$1 - \tau(\zeta) = 1 - \frac{1}{1 + e^{-\zeta}} = \frac{e^{-\zeta}}{1 + e^{-\zeta}} =: \sigma(-\zeta), \quad (5.3)$$

which may be used to write the gradient of the logistic function from (5.2) as

$$\nabla f(x) = -\frac{1}{N} \sum_{\ell=1}^N [1 - \tau(y_{\ell} d_{\ell}^T x)] y_{\ell} d_{\ell} = -\frac{1}{N} \sum_{\ell=1}^N \sigma(-y_{\ell} d_{\ell}^T x) y_{\ell} d_{\ell} = -\frac{1}{N} (v(x)^T D)^T$$

with

$$[v(x)]_\ell := \sigma(-y_\ell d_\ell^T x) y_\ell.$$

To find the search direction \bar{d}_k in Step 10 of Algorithm 7, we often prefer to use the CG method, which requires matrix-vector products with a submatrix of the Hessian matrix. The second derivative of f as defined in (5.2) may be derived by letting $d_\ell(i)$ denote the i th element of d_ℓ , using $y_\ell^2 = 1$ because $y_\ell \in \{-1, 1\}$ for all $\ell \in \{1, 2, \dots, N\}$, and (5.3) to obtain

$$[\nabla^2 f(x)]_{ij} = \frac{d}{dx_j} [\nabla_i f(x)] = \frac{1}{N} \sum_{\ell=1}^N \sigma(-y_\ell d_\ell^T x) [1 - \sigma(-y_\ell d_\ell^T x)] d_\ell(j) d_\ell(i),$$

from which it follows that the second-derivative matrix may be written as

$$\nabla^2 f(x) = (1/N) D^T E(x) D, \quad (5.4)$$

where $E(x)$ is a diagonal matrix with entries

$$[E(x)]_{\ell\ell} := \sigma(-y_\ell d_\ell^T x) [1 - \sigma(-y_\ell d_\ell^T x)] \quad \text{for all } \ell \in \{1, 2, \dots, N\}.$$

Using (5.4) and the definition of the matrix H_k in Step 8 of Algorithm 7, it follows that the required Hessian-vector products take the form

$$H_k s = [\nabla_{\mathcal{I}_k \mathcal{I}_k}^2 f(x_k)] s = (1/N) D(:, \mathcal{I}_k)^T E(x) D(:, \mathcal{I}_k) s = (1/N) (w(x)^T D(:, \mathcal{I}_k))^T,$$

where

$$[w(x)]_\ell := [E(x)]_{\ell\ell} [D(:, \mathcal{I}_k) s]_\ell.$$

Therefore, each Hessian-vector product proceeds by first multiplying s by a subset of the data matrix (namely, $D(:, \mathcal{I}_k)$), followed by N scalar multiplications

to obtain the vector $w(x)$, followed by inner-products of $w(x)$ with $|\mathcal{I}_k|$ columns from the data matrix, and a final scaling by $1/N$. It follows that the overall efficiency of computing each Hessian-vector product depends on the size of the set \mathcal{I}_k and the sparsity of the data matrix D .

Another way to find the vector \bar{d}_k in Step 10 of Algorithm 7 is to use a CD method for approximately minimizing the model $m_k(d) = g_k^T d + \frac{1}{2} d^T H_k d$, where $g_k = \nabla_{\mathcal{I}_k} F(x_k)$, $H_k = \nabla_{\mathcal{I}_k \mathcal{I}_k}^2 F(x_k) \equiv \nabla_{\mathcal{I}_k \mathcal{I}_k}^2 f(x_k)$, and $\mathcal{I}_k = \{i_1, i_2, \dots, i_{|\mathcal{I}_k|}\}$ contains indices for some subset of variables that are non-zero at x_k . In particular, for the k th iteration, such a strategy computes a sequence of sub-iterates $\bar{d}_{k,j}$ with $\bar{d}_{k,0} = 0$ via the update

$$d_{k,j+1} = d_{k,j} + \alpha_j e_{i_j},$$

where $i_j \in \mathcal{I}_k$ and

$$\alpha_j := \operatorname{argmin}_{\alpha \in \mathbb{R}} m_k(\bar{d}_{k,j} + \alpha e_j).$$

Setting the derivative of the objective equal to zero and solving for α , we get

$$\alpha_j = -\frac{(g_j + d_{k,j}^T H_k(:, j))}{[H_k]_{jj}}.$$

Notice that the j th diagonal element of H_k may be computed via

$$[H_k]_{jj} = [\nabla^2 f(x_k)]_{i_j i_j} = (1/N) F(:, i_j)^T D(x) F(:, i_j),$$

which means that the diagonal elements that correspond to the set \mathcal{I}_k may be computed in advance and then used in computing α_j when needed. Moreover,

the quantity $d_{k,j}^T H_k(:, j)$ may be computed as

$$\begin{aligned}
d_{k,j}^T H_k(:, j) &= H_k(:, j)^T d_{k,j} \\
&= (1/N) \left(F(:, \mathcal{I}_k)^T D(x) F(:, i_j) \right)^T d_{k,j} \\
&= (1/N) F(:, i_j)^T D(x) F(:, \mathcal{I}_k) d_{k,j},
\end{aligned}$$

which can be efficiently computed by updating $(1/N) D(x) F(:, \mathcal{I}_k) d_{k,j}$ via

$$\begin{aligned}
(1/N) D(x) F(:, \mathcal{I}_k) d_{k,j+1} &= (1/N) D(x) F(:, \mathcal{I}_k) (d_{k,j} + \alpha_j e_j) \\
&= (1/N) D(x) F(:, \mathcal{I}_k) d_{k,j} + \alpha_j (1/N) D(x) F(:, i_j).
\end{aligned} \tag{5.5}$$

We also need the residual of the associated linear system in order to decide termination of the CD iterations, which may be updated via

$$\begin{aligned}
r_{k,j+1} &= H_k d_{k,j+1} + g_k = H_k (d_{k,j} + \alpha_j e_j) + g_k = H_k d_{k,j} + g_k + \alpha_j H_k e_j \\
&= r_{k,j} + \alpha_j H_k(:, j) = r_{k,j} + \alpha_j [\nabla^2 f(x_k)](\mathcal{I}_k, i_j) \\
&= r_{k,j} + \alpha_j (1/N) F(:, \mathcal{I}_k)^T D(x) F(:, i_j) \\
&= r_{k,j} + \alpha_j (1/N) [D(x) F(:, \mathcal{I}_k)]^T F(:, i_j) \\
&= r_{k,j} + \alpha_j (1/N) \left(F(:, i_j)^T [D(x) F(:, \mathcal{I}_k)] \right)^T.
\end{aligned}$$

We can also observe that

$$r_{k,j+1} = H_k d_{k,j+1} + g_k = (1/N) F(:, \mathcal{I}_k)^T D(x) F(:, \mathcal{I}_k) d_{k,j+1} + g_k,$$

which is the same computation needed for one CG iteration, but if we use the fact that we also computed part of it in (5.5), it is then cheaper than CG.

5.1.2.3 Least-squares loss

The least-square objective function defined in (2.10) is commonly used to approximate the solution of an overdetermined system, i.e., sets of equations in which there are more equations than unknowns. For example, it is used in data fitting and regression where A represents the data design matrix D and the vector b represents the class label vector y . The corresponding ℓ_1 -regularization problem is often called the Lasso problem, which specifically is written as

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \left\{ f(x) + \lambda \|x\|_1 \equiv \frac{1}{2} \|Dx - y\|_2^2 + \lambda \|x\|_1 \right\} \quad (5.6)$$

where as usual $\lambda > 0$ is a weighting parameter.

Compared with the logistic objective function discussed in Section 5.1.2.2, the least-square objective function in problem (5.6) has a simpler form for the gradient vector and Hessian matrix, namely

$$g(x) = D^T(Dx - y) \quad \text{and} \quad (5.7a)$$

$$H = H(x) = D^T D. \quad (5.7b)$$

Notice that the Hessian matrix (5.7b) associated with the least-squares objective function is independent of x . In the context of FaRSA, this means that for any index set $\mathcal{I}_k \subseteq \mathcal{I} := \{1, 2, \dots, n\}$, we have that

$$H_{\mathcal{I}_k \mathcal{I}_k} = D_{\mathcal{I} \mathcal{I}_k}^T D_{\mathcal{I} \mathcal{I}_k}.$$

Based on the above, it is natural to ask whether pre-computing and saving $D^T D$ is a good idea, which may potentially reduce the computational cost. However, choosing to compute $H = D^T D$ in advance is not a trivial decision. In the FaRSA software, we use the following simple strategy to guide our decision.

The first factor to be considered is the memory consumption for saving H . If the memory requirement exceeds a threshold, computing H in advance should not be performed. If the threshold is not exceeded, then we should weigh the computational costs of the two competing options. Note that even though D may be a sparse matrix, the matrix $H = D^T D$ is rarely sparse. Hence we use a dense storage of the matrix H , which requires n^2 constants of type double. Subsequently, the memory consumption increases rapidly as n increases as shown in Figure 5.1. Since modern personal computers usually contain 8 to 32 GB of Random-access memory (RAM) (workstations or super computers likely possess more), we set the default threshold for possibly storing H to be $n = 10000$. In other words, if the number of variables in problem (5.6) is larger than 10000, then our FaRSA software will never pre-compute and store the full Hessian matrix H ; otherwise, FaRSA may choose to store H based on other factors that we discuss next.

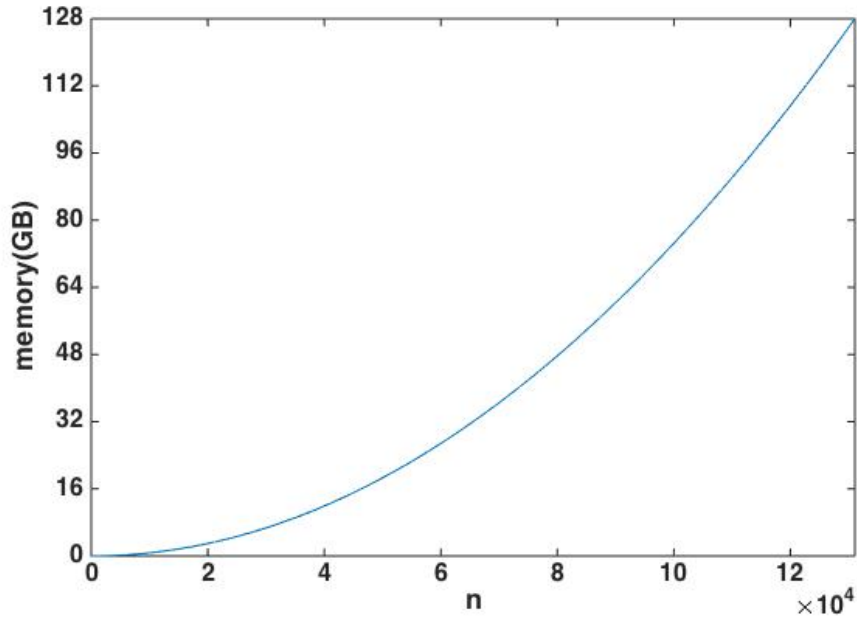


Figure 5.1: Required Memory (GB) for saving an n -by- n dense matrix of type double: $\text{mem}(\text{GB}) = \frac{8n^2}{1024^3}$. One double variable needs 8 bytes in memory.

If sufficient memory is available as described in the previous paragraph, then we should consider time complexity aspects. When objective-reducing steps are computed in FaRSA, the search direction \bar{d}_k in Step 10 of Algorithm 7 can be computed by approximately solving the subproblem

$$\underset{\bar{d} \in \mathbb{R}^{|\mathcal{I}_k|}}{\text{minimize}} \frac{1}{2} \|D_{\mathcal{I}\mathcal{I}_k} ([x_k]_{\mathcal{I}_k} + \bar{d}) - y\|^2 + \lambda \xi_k^T ([x_k]_{\mathcal{I}_k} + \bar{d}) \quad (5.8)$$

where $\xi_k \in \{-1, 0, +1\}^{|\mathcal{I}_k|}$. There exist many approaches for solving (5.8) exactly (e.g. using a Cholesky factorization, QR factorization, or Singular-value decomposition (SVD) [70]) or inexactly (e.g., using CG and CD). The most basic implementations that use the Cholesky and QR factorizations require the matrix $D_{\mathcal{I}\mathcal{I}_k}$ to have full column rank, which is equivalent to $H_{\mathcal{I}_k\mathcal{I}_k}$ being positive definite since $H = D^T D$. The SVD factorization approach is more stable and can easily handle the case when $H_{\mathcal{I}_k\mathcal{I}_k}$ is singular. However, all of them are relatively expensive to use because of their time complexities, e.g., $\mathcal{O}(|\mathcal{I}_k|^3)$ and $\mathcal{O}(N|\mathcal{I}_k|^2 + N^2|\mathcal{I}_k| + |\mathcal{I}_k|^3)$ for different implementations [71, Chapter 5].

Motivated by the high computational cost associated with exact methods as described in the previous paragraph, in our FaRSA software we allow the possibility of still using the CG method to inexactly solve (5.8) even when H is stored. In order to decide whether to store H , we must take the time complexities of both strategies into account, in particular the cost of computing Hessian-vector products. Before presenting the details of our strategy, let us introduce some notation. Denote the total number of non-zero entries in D as nnz . Let nnz_r be the array that records the number of non-zero entries for each row of D , e.g. $nnz_r[1] = 2$ indicates that there are two non-zero elements in the first row of D . Similarly, let nnz_c be the array that records the number of non-zero entries for each column of D . For convenience, we assume that $\mathcal{I}_k = \mathcal{I}$

for all k , namely that the subproblem (5.8) is solved over \mathbb{R}^n . Clearly, this will be an overestimate of the size of the subproblems on most realistic problems.

To compute $H = D^T D$ in advance, we need to compute the matrix-matrix multiplication between D transpose and itself a single time; since D is assumed to be stored in a sparse format, there are at least nnz multiplications and $\sum_{i=1}^n \max(0, nnz_c[i] - 1)$ additions, and at most $N \cdot nnz$ multiplications and $\sum_{i=1}^n N \cdot \max(0, nnz_c[i] - 1)$ additions. For each iteration in CG method, a matrix-vector multiplication between the Hessian matrix H in dense format and a dense vector v needs n^2 multiplications and $n^2 - n$ additions.

If $H = D^T D$ is not computed in advance, then for each iteration of CG the product of the Hessian matrix with a vector is achieved in the following way:

$$D^T(Dv) = Hv,$$

i.e., do a matrix-vector multiplication between D in sparse format and a dense vector v follows by a multiplication by D^T in sparse format with the computed dense vector Dv to obtain the final desired Hv . In general, this means that there will be $2nnz$ multiplications, and the number of additions will be

$$\sum_{i=1}^N \max(0, nnz_r[i] - 1) + \sum_{i=1}^n \max(0, nnz_c[i] - 1);$$

calculating Dv needs $\sum_{i=1}^N \max(0, nnz_r[i] - 1)$ additions, and then computing $D^T(Dv)$ needs $\sum_{i=1}^n \max(0, nnz_c[i] - 1)$ additions. We summarize the costs in memory and time for both strategies in Table 5.1.

Table 5.1: Resource requirements for when H is formed and saved (“Saving H ”) versus when it is used via matrix-vector products (“Not solving H ”).

	Task	Saving H	Not saving H
memory	$D^T D$	n^2 double variables	0
multiplications	$D^T D$	$N \cdot nnz$	0
additions	$D^T D$	$\sum_{i=1}^n N \max(0, nnz_c[i] - 1)$	0
multiplications	Hv	n^2	$2nnz$
additions	Hv	$n^2 - n$	$\sum_{i=1}^N \max(0, nnz_r[i] - 1)$ $+ \sum_{i=1}^n \max(0, nnz_c[i] - 1)$

For completing the consideration of time complexity, we are left to figure out the cost of doing multiplications and additions between two double variables. In general, double multiplication is more costly compared to double addition, but there is no certain answer to how much slower since it depends on many external factors, e.g. CPU architecture, digit circuit, and operational amplifier [72]. Here, we provide an empirical result by testing 10^9 random double-variable multiplications and additions in C programming language, which shows that double multiplication is roughly two times slower than double addition. Thus, we can define a computation score to measure the cost of the strategies:

cost(save = yes)

$$\begin{aligned}
&= T \frac{2}{3} n^2 + T \frac{1}{3} (n^2 - n) + \frac{2}{3} \frac{N+1}{2} nnz + \frac{1}{3} \sum_{i=1}^n \frac{N+1}{2} \max(0, nnz_c[i] - 1) \\
&= T n^2 - \frac{1}{3} T n + \frac{N+1}{3} nnz + \frac{N+1}{6} \sum_{i=1}^n \max(0, nnz_c[i] - 1)
\end{aligned} \tag{5.9a}$$

cost(save = no)

$$\begin{aligned}
&= T \frac{2}{3} 2nnz + T \frac{1}{3} \left[\sum_{i=1}^N \max(0, nnz_r[i] - 1) + \sum_{i=1}^n \max(0, nnz_c[i] - 1) \right] \\
&= \frac{4}{3} T \cdot nnz + \frac{1}{3} T \left[\sum_{i=1}^N \max(0, nnz_r[i] - 1) + \sum_{i=1}^n \max(0, nnz_c[i] - 1) \right]
\end{aligned} \tag{5.9b}$$

where we estimate the number of multiplications and additions for the single

time calculating $D^T D$ by the average quantities, and $T \geq 0$ is an estimate of the number of required Hessian-vector multiplications.

We now state the whole strategy for deciding whether to pre-computing H Algorithm 8. As shown in Algorithm 8, we first check whether there is enough memory to save H . If the dimension of H is too large, then a non-saving strategy is selected in line 3. If the dimension of H is small enough, then the computational cost is considered. The computation cost score for both strategies are calculated by (5.9a) and (5.9b) in line 5. The strategy with the smaller computational cost score is selected between line 6 and line 9. In FaRSA, we set default values as $n_t = 10000$ and $T = 10$.

Algorithm 8 Branch between saving H strategy and not saving H strategy

```

1: Input: Set  $n_t$ , and  $T$ .
2: if  $n \geq n_t$  then
3:   Select non-saving strategy.
4: else
5:   Calculate cost(save = Yes) and cost(save = No) by (5.9a) and (5.9b).
6:   if cost(save = Yes)  $\geq$  cost(save = No) then
7:     Select non-saving strategy.
8:   else
9:     Select saving strategy.

```

5.2 Solvers Tested

The solvers we chose to test in this chapter mostly come from the popular solvers that we described in Chapter 3, and a few additional solvers. In this section, we present implementation information for all of the solvers tested.

- **FaRSA:** We have both a C and Matlab implementation of our FaRSA software based on Algorithm 7. The version of FaRSA tested in this thesis is version 2.0, which is an improved version with memory optimization from version

1.0 that we used in [52]. It is currently available at the following webpage:

<https://github.com/daniel-p-robinson/FaRSA>

- **LIBLINEAR:** The LIBLINEAR solver [18] is an implementation of a proximal-Newton method that uses coordinate block-wise minimization to approximately solve its subproblems. We downloaded the latest version of LIBLINEAR, i.e. version 2.1. Instead of running LIBLINEAR through one of its wrappers of other high-level programming languages, we used directly its C/C++ program for fairness. The download url for their code is as follows:

<https://www.csie.ntu.edu.tw/~cjlin/liblinear>

- **ASACG:** The ASACG [38] method is a successful bound-constrained optimization method that we use to solve the smooth bound-constrained reformulation of problem (1.1). Specifically, we use ASACG to solve the problem

$$\underset{u \in \mathbb{R}^n, v \in \mathbb{R}^n}{\text{minimize}} \quad f(u - v) + \lambda e^T(u + v) \quad \text{subject to} \quad u \geq 0, v \geq 0$$

as described in Section 3.3. The version that we use is a C implementation that was downloaded from the following author’s homepage:

http://users.clas.ufl.edu/hager/papers/CG/Archive/ASA_CG-3.0.tar.gz

We also comment that ASACG allows for generic objective functions f , which requires users to implement at least two subroutines.

- **OWL-QN:** The version that we used is a C implementation created by the authors of OWL-QN from the Microsoft store:

<https://www.microsoft.com/en-us/download/confirmation.aspx?id=52452>

- **OBA:** We downloaded the latest version of OBA from

<https://github.com/keskarnitish/OBA>

At the time of this thesis, only a Matlab version of OBA was available.

- **SPAMS:** Sparse Modeling Software (SPAMS) is a C++ implementation of a proximal method, e.g. ISTA and FISTA with a Matlab interface for solving a large class of sparse approximation problems with different combinations of loss and regularizations. One of the main features of this toolbox is to provide a robust stopping criterion based on duality gaps to control the quality of the optimization, whenever possible. It also handles sparse feature matrices for large-scale problems. Since FISTA has been shown to outperform ISTA in general, we only test the FISTA routine of SPAMS:

<http://spams-devel.gforge.inria.fr>

- **IRPN:** The inexact regularized proximal Newton (IRPN) method is a family of sequential quadratic approximation (SQA) methods to solve problem (1.1), and proposed in [73]. The IRPN software was kindly provided by the author Man-Chung by request, which is a MATLAB implementation.

- **L1General:** L1General is from the University of British Columbia and contains a set of Matlab routines for various solvers for ℓ_1 -regularization problems. The software may be downloaded at the following webpage:

<https://www.cs.ubc.ca/~schmidtm/Software/L1General.html>

By considering the numerical results in [74, 75, 76], we chose the four best methods in L1General: Shooting [13], Gauss-Seidel [77], PSSgb [74], and PSSas [74].

5.3 Experimental Setup

The test problem is an ℓ_1 -regularized logistic regression problems of the form (1.1) for binary classification with f defined in (1.6) and $\lambda = 1/N$ the weighting pa-

parameter in (1.1). Such problems arise in the context of model prediction, making the design of advanced optimization algorithms that efficiently solve them paramount in big data applications. Furthermore, all test solvers described in Section 5.2 have special routines for the logistic regression objective function, except for ASACG which required us to implement interfaces. Our experiments were conducted using the cluster at the Computational Optimization Research laboratory at Lehigh University (COR@L) with a single-core 2.0GHz AMD CPU and 30GB of main memory. The selected tolerance for all solvers is $\epsilon = 10^{-6}$.

5.4 Data Set Description

We tested the solvers on numerous realistic data sets from the well-known LIBSVM repository (<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>). We selected the data sets by the following criterion. At first, we excluded all problems with multiple-classes (greater than two) so that each test problem is a binary classification problem. The remaining datasets were binary classification examples from which we removed *webspam* since insufficient computer memory was available. (All experiments were conducted on a 64-bit machine with 30GB of main memory.) Finally, for the adult data (*a1a*–*a9a*) and webpage data (*w1a*–*w8a*), we only used the largest instances, namely problems *a9a* and *w8a*. This left us with our final set of datasets from LIBSVM.

We now briefly describe the datasets that were chosen as described in the previous paragraph. One can observe that the collection is quite diverse.

- **a9a:** The *a9a* dataset is a subset of the *Adult* dataset from the UCI machine learning repository. It contains 123 binary variables describing properties of 32,561 website samples that have been tested against the adult website. The density of the data is 11.28%, where density refers to the percentage of nonzero

entries in the data matrix D . It is now available from LIBSVM repository:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/a9a>

- **australian:** This dataset about Australian credit approval is used for auditing credit card applications. It contains 14 variables that correspond to 6 numerical and 8 categorical attributes. This dataset also comes from the UCI machine learning repository. The LIBSVM repository provides its scaled version for which values of all attributes are scaled into $[-1, 1]$. The scaled *australian* has density 87.443%, and is available at the following webpage:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/australian_scale

- **avazu-app.tr:** This dataset was used in a competition on click-through rate prediction that was jointly hosted by Avazu and Kaggle in 2014 [78]. The participants were asked to learn a model from the first 10 days of an advertising log, and predict the click probability for the impressions on the 11th day. LIBSVM was the winner for this competition and generated the *avazu-app* dataset from its winning model [79]. The generated *avazu-app* dataset contains 999,990 variables to describe 12,642,186 samples as training data, and 4,577,464 samples as testing data. The density of the training *avazu-app.tr* dataset is 0.002%, indicating that it is very sparse. The training and testing *avazu-app* datasets are accessible at their respective webpages:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/avazu-app.tr.bz2>

and

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/avazu-app.t.bz2>

- **breast-cancer:** The *breast-cancer* dataset was originally created by Dr. William H. Wolberg for the purpose of breast-cancer diagnosis [80], and has since been preprocessed by LIBSVM. Currently, there exists 10 positive integer variables to describe the clinical report of 683 people. The density of the *breast-cancer* dataset is 100%, which is available at the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/breast-cancer>

- **cod-rna:** The *cod-rna* dataset was studied in [81] and involves using the secondary structure formation free energy change to generate 8 features describing 59,535 RNAs, and then used to detect non-coding instances. Besides the 59,535 RNA samples, an additional 271,617 samples are available as a validation set. The 8 variables consist of one negative integer feature, one positive integer feature for the RNA length, and six features scaled into the interval $[0, 1]$. The training dataset is 100% dense, and available at the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/cod-rna>

The validating data is available at the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/cod-rna.t>

- **colon-cancer:** The source data is created in [82] and concerns colon cancer diagnosis. The version we used has been preprocessed by LIBSVM after instance-wise normalization to mean zero and variance one, and then followed by feature-wise normalization to mean zero and variance one. The data has 62 variables and 2,000 samples with density 100%, and can be downloaded from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/colon-cancer.bz2>

- **covtype.binary:** This data is used for predicting the forest cover type by using cartographic variables only (no remotely sensed data). It is available from the UCI machine learning repository, and contains 54 integer attributes for describing 581,012 forests with density 21.998%. It is accessible at the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/covtype.libsvm.binary.bz2>

- **diabetes:** The *diabetes* dataset contains 8 positive variables to describe the properties of 768 diabetes patient records. The density of this dataset is 100%, and it is available at the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/diabetes>

- **duke-breast-cancer:** This data set was studied in [83] for breast-cancer diagnosis by gene expression and was preprocessed by [77]. It contains 7,129 gene expression features for 38 training and 4 validating instances with instance-wise normalization to mean zero and variance one. The resulting training dataset has a density of 100% and is available at the webpage

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/duke.tr.bz2>

The validation set can be obtained from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/duke.val.bz2>

- **fourclass:** This dataset was originally created in [84] for a four-class classification, and was since transformed into a binary classification problem by

LIBSVM. The dataset *fourclass* includes 2 integer variables and 862 samples that is 100% dense, and can be accessed from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/fourclass>

- **epsilon:** The *epsilon* dataset comes from the PASCAL Challenge 2008 and is split into two parts: 4/5 is used for training and 1/5 is used for testing. The training set is preprocessed using feature-wise normalization to mean zero and variance one and then instance-wise scaled to have unit length. Using the scaling factors of the training data, the testing data is processed in a similar way. In Total, there are 400,000 training instances and 100,000 testing instances, each with 2,000 features. The density for the training dataset is 100%. The training data is available at the following webpage:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/epsilon_normalized.bz2

The testing data may be obtained from the following webpage:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/epsilon_normalized.t.bz2

- **german numer:** The *german numer* dataset is a study of classifying people who are described by a set of attributes as either good or bad credit risk. There are 24 categorical integer attributes and 1,000 instances. The density of the data set is 100%, and it can be obtained from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/german.numer>

- **heart scale:** This dataset consists of heart disease data that includes 13 attributes scaled into the interval $[-1, 1]$, 270 samples, and a density of 96.24%. It is available at the following webpage:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/heart_scale

- **ionosphere:** This radar dataset was collected from a system in Goose Bay, Labrador [85]. “Good” radar returns are those showing evidence of some type of structure in the ionosphere, while “Bad” returns are those that do not pass through the ionosphere. There are 351 instances and 17 pulse numbers for the Goose Bay system, and each pulse number contains two attributes. Thus, the *ionosphere* dataset contains 351 instances with 34 features with a density of 88.41%. A variant that has been scaled into $[-1, 1]$ is available from the following webpage:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/ionosphere_scale

- **HIGGS:** The *HIGGS* dataset is for a classification problem that distinguishes between a signal process that produces Higgs bosons and a background process that does not. It possesses 28 real attributes, 11,000,000 data points, and has 92.112% density. It was downloaded from the webpage

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/HIGGS.bz2>

- **ijcnn1:** This dataset was used in the IJCNN 2001 neural network competition and then further transformed using the winner’s transformation. The *ijcnn1* dataset contains 22 features with 49,990 instances for training and 91,701 instances for testing. The data matrix has density 59.09%. The training and testing datasets can be downloaded from the following webpages:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/ijcnn1.tr.bz2>

and

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/ijcnn1.t.bz2>

- **kdda**: This data comes from Carnegie Learning and DataShop [86], and is the training set for the first problem in the KDD Cup 2010, which is an educational data mining competition. It is a high-dimensional sparse data set with 8,407,752 instances and 20,216,830 features and a density of 0.0002%.

The training and testing datasets may be downloaded from the webpages

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/kdda.bz2>

and

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/kdda.t.bz2>

- **kddb**: Similar to *kdda*, the *kddb* dataset also comes from the Carnegie Learning and DataShop [86], and is the training set for the second problem in the KDD Cup 2010. It is a high-dimensional sparse data that consists of 19,264,097 samples, 29,890,095 attributes, and has a density of 0.0001%. The training data is available at the webpage

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/kddb.bz2>

while the testing dataset is available at

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/kddb.t.bz2>

- **leukemia**: The *leukemia* dataset is another gene expression dataset for cancer diagnosis [87] with instance-wise and feature-wise normalization to mean zero and variance one (the density is 100%). It is split into a training set of 38 instances and a testing set of 44 instances. The number of attributes is

7,129. These training dataset can be obtained from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/leu.bz2>

while the testing dataset can be obtained from

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/leu.t.bz2>

- **liver-disorder:** The *liver-disorder* dataset comes from the UCI machine learning repository but has been preprocessed by LIBSVM. It possesses 345 instances with six features, resulting in a density of 100%. It can be downloaded from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/liver-disorders>

- **madelon:** This dataset is from the NIPS 2003 Feature Selection Challenge. It contains 500 features, 2000 training samples (with density of 100%), and 600 testing samples. They can be downloaded from the following webpages:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/madelon>

and

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/madelon.t>

- **mushrooms:** The *mushrooms* dataset contains 112 binary attributes for describing the physical characteristics used to classify whether a mushroom is poisonous or edible. There is a total of 8,124 samples. The dataset can be downloaded from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/mushrooms>

- **news20.binary:** This data is studied in [88]. It is a sparse dataset with density 0.034% that has 19,996 instances and 1,355,191 positive real-valued features. For each instance, the summation of its features is equal to one. The dataset may be downloaded from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/news20.binary.bz2>

- **phishing:** The *phishing* dataset comes from the UCI Machine Learning repository and has been preprocessed by LIBSVM so that each feature vector is normalized to maintain unit-length [89]. In total, there are 68 features and 11,055 examples with density 44.12%. It can be obtained from the webpage

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/phishing>

- **rcv1.binary:** This is a well-known text categorization dataset used in [90]. It is sparse with a density of 0.157%. There exist 47,236 variables and 20,242 training examples, which may be downloaded at the following webpage:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/rcv1_train.binary.bz2

In addition, rcv1.binary has a much larger testing set with 677,399 instances, which can be obtained at the following webpage:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/rcv1_test.binary.bz2

- **real-sim:** This dataset is associated with the classification of real versus simulation data. It contains 72,309 samples, 20,958 features, and has a density of 0.25%. It can be obtained from the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/real-sim.bz2>

- **skin_nonskin:** The *skin_nonskin* dataset contains skin texture information from faces of people of various ages, different genders, and various races. It contains 4 real-valued attributes and 245,057 instances with a density of 98.27%. It may be downloaded from the following webpage:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/skin_nonskin

- **sonar_scale:** This data studied the discrimination between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. It has 60 features scaled into $[-1, 1]$ and 208 samples, and a 99.992% density. It is now available at

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/sonar_scale

- **splice_scale:** This dataset is used to test the classification of two classes of splice junctions in a DNA sequence. It is split into 1,000 training and 2,175 testing instances with 60 attributes scaled into the interval $[-1, 1]$. The training and test sets can be downloaded from

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/splice_scale

and

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/splice.t>

- **SUSY:** The *SUSY* dataset is used to test the whether a signal process produces supersymmetric particles or not. It contains 5,000,000 data examples, each with 18 real-valued features. The density of the SUSY data set is 98.82%. It can be found at the following webpage:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

[binary/SUSY.bz2](#)

- **svm_guide1:** This dataset was created for an astro-particle application and preprocessed by LIBSVM. It contains a training set with 3,089 instances and a testing set with 4,000 instances, each with 4 real features and a density level of 99.70%. These two sets are available at the following webpages:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

[binary/svmguide1](#)

and

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

[binary/svmguide1.t](#)

- **svmguide3:** The *svmguide3* dataset contains 21 variables that have been scaled into the interval $[-1, 1]$. It consists of 1,243 instances for training (density of 84.324%) and 41 instances for testing. They can be downloaded from the following webpages:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

[binary/svmguide3](#)

and

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

[binary/svmguide3.t](#)

- **url:** The *url* dataset is for identifying suspicious urls [91]. It is a high dimensional data set (3,231,961 variables and 2,396,130 samples), but is sparse with a density of 0.004%. it may be downloaded from the webpage

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

[binary/url_combined.bz2](#)

- **w8a:** The *w8a* dataset is another adult dataset with 49,749 training samples, 14,951 testing samples, and 300 binary features. It was studied in [92]

and has a density of 3.88%. The training and testing sets can, respectively, be accessed at the following webpages:

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/w8a>

and

<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/w8a.t>

The above 35 datasets are summarized in Table 5.2. Instead of listing the datasets alphabetically, they are presented in ascending order based on file size (MB). The seven datasets under the dashed line (i.e. *url_combined*, *SUSY*, *avazu-app.tr*, *kdda*, *kddb*, *HIGGS*, and *epsilon*) possess file sizes that exceed 2GB, and we refer them as “big datasets” in this thesis. The remaining data sets are referred to as “small datasets”.

5.5 Experimental Results

In this section, we test the algorithms listed in Section 5.2 in comparison to FaRSA so as to demonstrate its effectiveness numerically. The whole section can be partitioned into three parts. At first, we concentrate on comparing two main evaluation metrics of optimization algorithm, namely their run time and final objective function value. Other metrics, e.g. function evaluations and gradient evaluations, are also sometimes considered important but not all of the selected solvers track and output these quantities. Next, we discuss the local convergence properties of these solvers, and in particular present numerical results for FaRSA to verify the superlinear convergence rate established in Theorem 35. Finally, FaRSA is applied on a set of binary classification problems to further show its suitability on realistic and interesting applications.

Table 5.2: Set of binary classification datasets considered. They are organized in ascending order based on file size measured in megabytes (MB).

dataset	N	n	density(%)	size (MB)
liver-disorders	345	6	100.000	0.014
fourclass	862	2	100.000	0.024
heart	270	13	96.239	0.026
australian	690	14	87.443	0.068
diabetes	768	8	100.000	0.072
breast-cancer	683	10	100.000	0.078
ionosphere	351	34	88.411	0.102
svmguide1	3089	4	99.701	0.104
sonar	208	60	99.992	0.149
svmguide3	1243	22	84.335	0.165
german.numer	1000	24	100.000	0.273
splice	1000	60	100.000	0.517
mushrooms	8124	112	18.750	0.839
colon-cancer	62	2000	100.000	1.647
a9a	32561	123	11.276	2.222
phishing	11055	68	44.118	3.452
duke-breast-cancer	38	7129	100.000	3.721
leukemia	34	7129	100.000	3.723
cod-rna	59535	8	100.000	4.451
w8a	49749	300	3.883	4.926
skin-nonskin	245057	3	98.267	6.430
ijcnn1	49990	22	59.091	7.386
madelon	2000	500	100.000	7.428
rcv1.binary	20242	47236	0.157	34.85
gisette	6000	5000	12.971	47.40
covtype.binary	581012	54	21.998	59.58
real-sim	72309	20958	0.245	86.19
news20	19996	1355191	0.034	133.5
url_combined	2396130	3231961	0.004	2107
SUSY	5000000	18	98.820	2356
avazu-app.tr	12642186	999990	0.002	2361
kdda	8407752	20216830	0.0002	2546
kddb	19264097	29890095	0.0001	4894
HIGGS	11000000	28	92.112	7564
epsilon	400000	2000	100.000	11598

5.5.1 Runtime and final objective function value

This section focuses on comparing solvers in terms of computational time to solve the target problem and the final achieved objective function value. We aim for a fair comparison and thus try to eliminate the effect of the particular

programming language used. We group the tested solvers into two groups: one for C/C++ solvers and one for Matlab solvers. Then, we compare the C implementation of FaRSA with the solvers written in C/C++, and our Matlab version of FaRSA to the solvers written in Matlab. We note, however, that there are some algorithms such as IRPN that is primarily written in Matlab but uses mex files to accelerate the CD subproblem solver; we still consider it as a Matlab implementation. For the comparison with solvers written in Matlab, we only focus on the performance on small-scale test problems, since not all Matlab solvers are applicable in this setting. For the solvers written in C/C++, we compare them on both small-scale datasets and large-scale datasets.

5.5.1.1 Matlab versus C implementations of FaRSA

Let us compare our FaRSA implementations written in C and Matlab on the small-scale test problems in Table 5.3 with stopping tolerance of $\epsilon = 10^{-6}$; this test highlights the necessity for comparing the two groups of solvers separately. In order to obtain reliable timing information, we report under the column “Time (seconds)” the average CPU time over 10 trials for each solver on each dataset. The value of the objective function $F(x) = f(x) + \lambda\|x\|_1$ at the final iterate is stated under the column “F-final”, and the number of iterations is marked as “# of iter”. For each dataset the shortest required (average) computing time is noted with red font. As shown in Table 5.3, the C implementation of FaRSA outperforms our Matlab version on nearly all of these small-scale test problems. Both versions of FaRSA achieve the same final objective function values on most of the test problems (to five digits of accuracy). It is also notable that distinct programming languages may deliver different results for the same algorithm based on the number of iterations for these test problems.

Table 5.3: The required computing time, final objective function value, and the number of iterations for our C and Matlab implementations of FaRSA on the small-scale datasets listed in Table 5.2.

Dataset	FaRSA in C			FaRSA in Matlab		
	Time (secs)	F-final	# of iter	Time (secs)	F-final	# of iter
liver-disorders	0.0061	0.65047	8	0.0232	0.65047	8
fourclass	0.0053	0.53305	6	0.2093	0.53305	6
heart	0.0052	0.38025	7	0.0185	0.38025	8
australian	0.0138	0.33669	11	0.0395	0.33669	11
diabetes	0.0183	0.60913	17	0.0756	0.60913	22
breast-cancer	0.0132	0.15478	11	0.0242	0.15478	10
ionosphere	0.0122	0.37042	13	0.0475	0.37042	15
svmguide1	0.0144	0.32350	8	0.0511	0.32349	10
sonar	0.0168	0.47238	15	0.0654	0.47238	15
svmguide3	0.0296	0.50362	17	0.0740	0.50362	18
german.numer	0.0532	0.48005	24	0.1049	0.48005	24
splice	0.0159	0.50671	9	0.0457	0.50671	9
mushrooms	0.1045	0.00123	16	0.1781	0.00114	14
colon-cancer	0.0533	0.20047	48	0.3645	0.20047	48
a9a	2.5826	0.32428	28	3.1861	0.32428	36
phishing	0.2936	0.15890	14	0.3395	0.15890	19
duke-breast-cancer	0.1205	0.19326	19	0.4249	0.19326	28
leukemia	0.1736	0.17995	25	0.4425	0.17995	32
cod-rna	0.8449	0.19312	28	1.3650	0.19312	30
w8a	2.2587	0.12206	22	4.9653	0.12206	33
skin-nonskin	1.1958	0.00085	17	0.7681	0.00089	10
ijcnn1	0.7374	0.18461	10	0.2667	0.21391	9
madelon	22.1376	0.51685	51	36.0101	0.51685	167
rcv1.binary	1.3685	0.19252	14	2.74043	0.19252	17
gisette	18.3510	0.00012	66	13.77604	0.00013	40
covtype.binary	0.2550	0.00000	1	0.36832	0.00000	1
real-sim	8.1583	0.14125	16	10.2459	0.14125	18
news20	6.3979	0.32348	15	20.7303	0.32348	18

5.5.1.2 Comparison of C/C++ solvers

We first compare the performance of our C implementation of FaRSA with the state-of-the-art C/C++ solvers LIBLINEAR and ASACG on the datasets in Table 5.2 with stopping tolerance $\epsilon = 10^{-6}$. The results of our tests are presented in Table 5.4. In order to obtain reliable timing information, we report under the column “Time (seconds)” the average CPU time over 10 trials for each solver on each dataset. The value of the objective function $F(x) = f(x) + \lambda\|x\|_1$ at the final iterate is stated under the column “F-final”. For all three methods, we allowed a maximum of twelve hours of computing time. We indicate any

instance for which the twelve hours was reached before the termination condition was satisfied by writing “max time”, and use “mem” to indicate that the solver failed to solve the problem due to an out-of-memory error.

To facilitate a comparison among the three algorithms, for each dataset the shortest required computing time is noted with red font, the second shortest with blue font, and the longest with black font. Although the different solvers employ their own distinct termination conditions, they achieve the same final objective function values (to five digits of accuracy) on most of the datasets. Notable exceptions for which convergence for all methods occurs are datasets *breast-cancer*, *skin-nonskin*, *covtype.binary*, *leukemia*, *kdda*, *kddb*, and *epsilon* although they do not show any clear preference for one algorithm. Finally, although the maximum time limit is reached by LIBLINEAR and ASACG on datasets *avazu-app.tr* and *url_combined*, they still manage to find comparable final objective values, thus indicating that the termination condition plays an important role for these two datasets.

Table 5.4 shows that FaRSA generally performs better than LIBLINEAR and ASACG on these datasets. In fact, FaRSA achieves the best computational time on 19 datasets and achieves the second best computational time on 14 of the datasets. The second best performer is LIBLINEAR, which achieves the best computational time on 13 datasets and achieves the second best computational time on 8 of the datasets. In terms of reliability, FaRSA is the best since it succeeds on every dataset except *kdda* and *kddb* due to running out of time, whereas LIBLINEAR fails on four datasets, namely *avazu-app.tr*, *url_combined*, *kdda* and *kddb*, and ASACG fails on the same datasets with *epsilon* additionally because of running out of memory.

Next, we compare FaRSA with two other popular C/C++ solvers, namely FISTA implemented in SPAMS and OWL-QN on the same dataset col-

Table 5.4: The required computing time and final objective function value for our C implementation of FaRSA, LIBLINEAR, and ASACG on the datasets listed in Table 5.2.

Dataset	Time (seconds)			F-final		
	FaRSA	LIBLINEAR	ASACG	FaRSA	LIBLINEAR	ASACG
liver-disorders	0.0061	0.0086	0.0086	0.65047	0.65047	0.65047
fourclass	0.0053	0.0093	0.0083	0.53305	0.53305	0.53305
heart	0.0052	0.0037	0.0073	0.38025	0.38025	0.38025
australian	0.0138	0.0336	0.0162	0.33669	0.33669	0.33669
diabetes	0.0183	0.0139	0.1416	0.60913	0.60913	0.60913
breast-cancer	0.0132	0.0074	0.0712	0.15478	0.14626	0.14626
ionosphere	0.0122	0.0116	0.0157	0.37042	0.37042	0.37042
svmguide1	0.0144	0.0197	0.0836	0.32350	0.32350	0.32350
sonar	0.0168	0.0193	0.0314	0.47238	0.47238	0.47238
svmguide3	0.0296	0.0324	0.1065	0.50362	0.50362	0.50362
german.numer	0.0532	0.0351	0.3066	0.48005	0.48005	0.48005
splice	0.0159	0.0110	0.0187	0.50671	0.50671	0.50671
mushrooms	0.1045	0.1141	0.2377	0.00123	0.01014	0.00123
colon-cancer	0.0533	0.0214	0.1908	0.20047	0.20047	0.20047
a9a	2.5826	20.8732	18.2233	0.32428	0.32428	0.32428
phishing	0.2936	0.7855	1.8246	0.15890	0.15890	0.15890
duke-breast-cancer	0.1205	0.0621	0.6071	0.19326	0.19326	0.19326
leukemia	0.1736	0.0638	0.6760	0.17995	0.20112	0.17995
cod-rna	0.8449	10.4215	3.5237	0.19312	0.19312	0.19312
w8a	2.2587	1.8244	9.0901	0.12206	0.12206	0.12206
skin-nonskin	1.1958	4.3514	0.7119	0.00085	0.35153	0.00079
ijcnn1	0.7374	1.2669	3.2924	0.18461	0.18461	0.18461
madelon	22.1376	30496	903.67	0.51685	0.51685	0.51685
rcv1.binary	1.3685	0.9301	7.7141	0.19252	0.19252	0.19252
gisette	18.3510	2.8383	284.72	0.00012	0.00015	0.00012
covtype.binary	0.2550	9767	7.9744	0.00000	0.51367	0.00000
real-sim	8.1583	3.8144	19.2361	0.14125	0.14125	0.14126
news20	6.3979	13.9160	119.28	0.32348	0.32348	0.32348
url.combined	20613	max time	max time	0.02624	0.01812	0.03203
SUSY	317.93	15373	2744	0.46300	0.46300	0.46300
avazu-app.tr	25214	max time	max time	0.30009	0.30011	0.30071
kdda	max time	max time	max time	0.26441	0.26422	0.28475
kddb	max time	max time	max time	0.25129	0.25129	0.35451
HIGGS	623.39	11641	5022	0.63771	0.63771	0.63771
epsilon	1978.9	6794.3	mem	0.25162	0.25845	—

lection in Table 5.2 with the same tolerance $\epsilon = 10^{-6}$. The results of our tests are presented in Table 5.5 with the same evaluation measurements used in Table 5.4 where “max iter” indicates that the solver failed to solve the problem due to running out of the maximum allowed quantity of iterations. As the results show, FaRSA performs the best on

all of the test problems. FISTA reaches the maximum allowed number of iterations of 10^6 on problems *svmguide1*, *cod-rna*, *breast-cancer*, *skin-nonskin*, *german.numer*, *diabetes*, *madelon*, *a9a*, *liver-disorders*, *ijcnn1*, *covtype.binary*, *leukemia*, and *phishing*. FISTA exceeds the maximum allowed wall time for problems *gisette*, *covtype.binary*, *news20*, *avazu-app.tr*, *HIGGS*, *url.combined*, *SUSY*, *kdda*, *kddb*, and *epsilon*. Even though FISTA fails to solve the above test problems, it is worth noting that FISTA can achieve competitive final objective function values to FaRSA. In terms of reliability, FaRSA is the best since it succeeds on every dataset except *kdda* and *kddb* due to running out of time, whereas OWL-QN fails on 10 of the 35 test problems, and FISTA fails on even more. Another observation to be pointed out is that OWL-QN achieves less accurate final objective function values than FaRSA and FISTA on a majority of the datasets.

5.5.1.3 Comparison of Matlab solvers

In this section we compare our Matlab implementation of FaRSA with the solvers implemented in Matlab. As in the previous section, we break this comparison up into a couple of groups to make the comparison easier.

First, let us compare FaRSA with OBA and IRPN. The results of our tests are presented in Table 5.6 with the same evaluation measurements used in Table 5.4 and 5.5. In the tables, “bug” denotes that the solver operates or exits abnormally, i.e. it stops running due to some fatal error or returns a negative objective function, which is impossible for our selected target problem; we use “ascent” to denote that the objective function increased. In theory, ascent is only possible for OBA when their estimate (10^8 in their code) of the Lipschitz constant for the gradient of f is not large enough. Although simple adaptive strategies could be used to avoid such issues, we made no such attempts be-

Table 5.5: The required computing time and final objective function value for our C implementation of FaRSA, FISTA (SPAMS), and OWL-QN on the datasets listed in Table 5.2.

Dataset	Time (seconds)			F-final		
	FaRSA	FISTA	OWL-QN	FaRSA	FISTA	OWL-QN
liver-disorders	0.0061	max iter	0.2877	0.65047	0.65048	0.65336
fourclass	0.0053	0.5949	0.0865	0.53305	0.53305	0.53421
heart	0.0052	0.2065	0.1050	0.38025	0.38025	0.38396
australian	0.0138	1.3491	1.1196	0.33669	0.33669	0.33814
diabetes	0.0183	max iter	2.1602	0.60913	0.60915	0.61043
breast-cancer	0.0132	max iter	0.7931	0.15478	0.18652	0.14773
ionosphere	0.0122	2.7880	1.2863	0.37042	0.37042	0.37328
svmguide1	0.0144	max iter	1.0645	0.32350	0.32388	0.32383
sonar	0.0168	3.7056	5.9507	0.47238	0.47238	0.47721
svmguide3	0.0296	9.5226	4.4579	0.50362	0.50407	0.50443
german.numer	0.0512	max iter	10.2895	0.48005	0.48006	0.48106
splice	0.0159	0.2054	1.2725	0.50671	0.50671	0.50771
mushrooms	0.1045	5.7506	22.2304	0.00123	0.00130	0.00135
colon-cancer	0.0533	25.6793	16.5335	0.20047	0.20047	0.21660
a9a	2.5826	max iter	139.21	0.32428	0.32428	0.32441
phishing	0.2936	max iter	112.84	0.15890	0.15890	0.15899
duke-breast-cancer	0.1205	46.5336	54.9510	0.19326	0.19326	0.21958
leukemia	0.1736	max iter	76.1660	0.17995	0.17995	0.20627
cod-rna	0.8449	max iter	537.58	0.19312	0.19389	0.19496
w8a	2.2587	39777	54.3395	0.12206	0.12207	0.12208
skin-nonskin	1.1958	max iter	74.4724	0.00085	0.00079	0.00079
ijcnn1	0.7374	max iter	32.9815	0.18461	0.21391	0.18463
madelon	22.1376	max iter	max time	0.51685	0.51685	0.58430
rcv1.binary	1.3685	40647	116.54	0.19252	0.19253	0.19257
gisette	18.3510	max time	max time	0.00012	0.00012	0.00012
covtype.binary	0.2550	max time	max time	0.00000	0.00000	0.0001
real-sim	8.1583	38738	217.84	0.14125	0.14126	0.14127
news20	6.3979	max time	1063.41	0.32348	0.32349	0.32353
url_combined	20613	max time	max time	0.02624	0.03860	0.0294
SUSY	317.93	max time	max time	0.46300	0.46301	0.46680
avazu-app.tr	25214	max time	max time	0.30009	0.30058	0.30052
kdda	max time	max time	max time	0.26441	0.26667	–
kddb	max time	max time	mem	0.25129	0.26596	–
HIGGS	666.98	max time	max time	0.63771	0.63771	0.65074
epsilon	1978.9	max time	max time	0.25162	0.25849	0.27018

cause we did not want to make any edit to their code.

Table 5.6 shows that FaRSA performs better than OBA and IRPN on most of these datasets. In fact, FaRSA achieves the best computational time on 20 datasets and achieves the second best computational time on 5 of the datasets. The second best performer is OBA, which achieves the best computational

time on 5 datasets and achieves the second best computational time on 13 of the datasets. In terms of reliability, FaRSA is the best since it succeeds on every dataset, whereas OBA fails on the two datasets *breast-cancer* and *cov-type.binary*, and IRPN fails on more. IRPN performs well on some of the test problems, but the implementation of IRPN appears to contain some unknown bug that causes its failure on problems *skin-nonskin*, *mushrooms*, and *cov-type.binary*. We did not attempt to fix these bugs since we did not want to edit their source code. In terms of the final objective function values obtained, for the problems that IRPN successfully solved, it achieved the same final objective function values on most of them (an exception was problem *madelon*), and for the problems that OBA successfully solved, it reached the same final objective function values with one exception, namely the dataset *skin-nonskin*.

Next, we compare our Matlab implementation of FaRSA with the four best approaches implemented within the L1General suite of routines, with a stopping tolerance of $\epsilon = 10^{-6}$. Table 5.7 presents the comparison with the methods PSSas and PSSgb, whereas Table 5.8 presents the comparison to the methods Gauss-Seidel and Shooting. As shown in Table 5.7, FaRSA performs better than PSSas, and PSSgb on most of the test problems. In fact, FaRSA achieves the best CPU time on 22 out of the 28 datasets, and has the second best CPU time on 6 datasets. The next best solver, namely PSSas, is the best solver on 6 datasets and the second best solver on 20 datasets. As for the final objective function values, all three algorithms computed the same values (to five digits of accuracy) on most datasets.

The other two methods, namely Gauss-Seidel and Shooting, do not perform competitively based on the results of Table 5.8. In fact, FaRSA requires the shortest computational time on all of test problems, and also achieves a smaller final objective function value on most of datasets. In Table 5.8, “bug” means

Table 5.6: The computing time and final objective values for our Matlab implementation of FaRSA, OBA, and IRPN on the datasets listed in Table 5.2.

Dataset	Time (seconds)			F-final		
	FaRSA	OBA	IRPN	FaRSA	OBA	IRPN
liver-disorders	0.0232	0.1104	0.0465	0.65047	0.65047	0.65047
fourclass	0.2093	1.3311	1.1877	0.53305	0.53305	0.53305
heart	0.0185	0.0516	0.0213	0.38025	0.38025	0.38025
australian	0.0395	0.0540	0.0838	0.33669	0.33669	0.33669
diabetes	0.0756	0.0610	0.3827	0.60913	0.60913	0.60913
breast-cancer	0.0242	ascent	0.5470	0.15478	–	0.14626
ionosphere	0.0122	0.26471	0.0525	0.37042	0.37042	0.37042
svmguide1	0.0511	0.0702	0.0853	0.32349	0.32349	0.32350
sonar	0.0654	0.1011	0.0858	0.47238	0.47238	0.47238
svmguide3	0.0740	0.1173	0.0883	0.50362	0.50407	0.50362
german.numer	0.1049	0.1282	0.5975	0.48005	0.48005	0.48005
splice	0.0457	0.0625	0.0408	0.50671	0.50671	0.50671
mushrooms	0.1781	0.2170	bug	0.00114	0.00114	–
colon-cancer	0.3645	0.1850	0.3455	0.20047	0.20047	0.20047
a9a	3.1861	9.5990	31.1678	0.32428	0.32428	0.32439
phishing	0.3395	0.39706	2.6990	0.15890	0.15890	0.15890
duke-breast-cancer	0.4249	0.4079	0.5900	0.19326	0.19326	0.19326
leukemia	0.4425	0.4039	0.6837	0.17995	0.17995	0.17995
cod-rna	1.3650	0.7439	27.8281	0.19312	0.19312	0.19312
w8a	4.9653	3.5904	2.7458	0.12206	0.12206	0.12206
skin-nonskin	0.7681	2.7555	bug	0.00089	0.00079	–
ijcnn1	0.2667	0.3063	1.8068	0.21391	0.21391	0.18461
madelon	36.0101	1530.50	50.5510	0.51685	0.51685	0.55383
rcv1.binary	2.7404	2.8199	7.9621	0.19252	0.19252	0.19252
gisette	13.7760	159.17	284.72	0.00013	0.00012	0.00012
covtype.binary	0.3683	ascent	bug	0.00000	–	–
real-sim	10.2459	8.7299	7.6075	0.14125	0.14125	0.14125
news20	20.7303	40.8101	185.39	0.32348	0.32348	0.32348

that the solver exited with an error in the code. More specifically, Gauss-Seidel and Shooting sometimes exited abnormally because of “Undefined function or variable ‘viol’”. We did not try to fix the bug since again we did not want to edit their source code. For test problems *gisette* and *real-sim*, Gauss-Seidel and Shooting reached the maximum allowed computing time of 12 hours with no objective function value returned since no iterations were completed.

Table 5.7: The computing time and final objective function value for our Matlab implementation of FaRSA, PSSas, and PSSgb on the datasets in Table 5.2.

Dataset	Time (seconds)			F-final		
	FaRSA	PSSas	PSSgb	FaRSA	PSSas	PSSgb
liver-disorders	0.0232	0.0243	0.2155	0.65047	0.65047	0.65047
fourclass	0.2093	1.0229	5.0579	0.53305	0.53305	0.53305
heart	0.0285	0.0245	0.0930	0.38025	0.38025	0.38025
australian	0.0395	0.0400	0.1531	0.33669	0.33669	0.33669
diabetes	0.0756	0.0626	0.2651	0.60913	0.60913	0.60913
breast-cancer	0.0242	0.0110	0.0469	0.15478	0.64068	0.14626
ionosphere	0.0475	1.15765	2.42222	0.37042	0.37042	0.37042
svmguide1	0.0511	0.0647	0.3228	0.32349	0.32349	0.32349
sonar	0.0654	0.1027	0.3268	0.47238	0.47238	0.47238
svmguide3	0.0740	0.0880	0.4005	0.50362	0.50407	0.50407
german.numer	0.1049	0.2466	0.7972	0.48005	0.48005	0.48005
splice	0.0457	0.0279	0.1149	0.50671	0.50671	0.50671
mushrooms	0.1781	0.7341	3.4040	0.00114	0.00114	0.00114
colon-cancer	0.3645	0.2518	0.8968	0.20047	0.20047	0.20047
a9a	3.1861	3.4612	11.2726	0.32428	0.32428	0.32428
phishing	0.3395	0.47533	2.17705	0.15890	0.15890	0.15890
duke-breast-cancer	0.4249	0.5668	1.9134	0.19326	0.19326	0.19326
leukemia	0.4425	0.6636	2.4772	0.17995	0.17995	0.17995
cod-rna	1.3650	1.6239	7.9827	0.19312	0.19312	0.19312
w8a	4.9653	5.7885	27.5056	0.12206	0.12206	0.12206
skin-nonskin	0.7681	0.6746	3.0623	0.00089	0.00079	0.00079
ijcnn1	0.2667	0.3473	1.6330	0.21391	0.21391	0.21391
madelon	36.0101	44.940	44.7294	0.51685	0.51685	0.54183
rcv1.binary	2.7404	5.3022	9.6716	0.19252	0.19252	0.19252
gisette	13.7760	149.90	205.93	0.00013	0.00012	0.00012
covtype.binary	0.3683	0.4207	1.5225	0.00000	0.00000	0.00000
real-sim	10.2459	27.0446	54.3067	0.14125	0.14125	0.14125
news20	20.7303	678.13	246.84	0.32348	0.32348	0.32348

5.5.2 Local convergence performance of FaRSA

In this section, we investigate the empirical local convergence of the iterates produced by FaRSA within the context predicted by Lemma 29(ii) and Theorem 35. In Table 5.9, we present the evolution of $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$ over the final 5 iterations. The column titled “final” gives the values for $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$ at the final iterate, the column titled “final-1” gives the values for $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$ at the penultimate iterate, and so on.

In Table 5.9 we can observe for the majority of data sets (23 out of 35) that the vector $\beta(x_k)$ is eventually equal to zero as predicted by Lemma 29(ii). At

Table 5.8: The computing time and final objective value for our Matlab implementation of FaRSA, Gauss-Seidel, and Shooting on the datasets in Table 5.2.

Dataset	Time (seconds)			F-final		
	FaRSA	Gauss-Seidel	Shooting	FaRSA	Gauss-Seidel	Shooting
liver-disorders	0.0232	10.0321	1.6633	0.65047	0.65047	0.65047
fourclass	0.2093	1.9887	1.4886	0.53305	0.53305	0.53305
heart	0.0285	4.7408	2.3444	0.38025	0.38025	0.38025
australian	0.0395	31.2889	21.4564	0.33669	0.33675	0.33669
diabetes	0.0756	44.2652	16.5659	0.60913	0.60996	0.60913
breast-cancer	0.0242	28.6877	8.9556	0.15478	0.15639	0.14626
ionosphere	0.0475	23.5422	15.86339	0.37042	0.37503	0.37042
svmguide1	0.0511	16.1192	4.7737	0.32349	0.32349	0.32349
sonar	0.0654	90.3944	45.3541	0.47238	0.49216	0.47360
svmguide3	0.0740	132.1301	64.9371	0.50362	0.50675	0.50407
german.numer	0.1049	114.6893	44.5434	0.48005	0.51218	0.48065
splice	0.0457	413.3176	75.4137	0.50671	0.50911	0.50671
mushrooms	0.1781	2.0501	199.29	0.00114	0.00114	0.00116
colon-cancer	0.3645	20399.21	2795.91	0.20047	0.21298	0.23788
a9a	3.1861	4059.53	2455.13	0.32428	0.34049	0.32486
phishing	0.3395	1047.13	710.06	0.15890	0.16609	0.16131
duke-breast-cancer	0.4249	max time	bug	0.19326	0.19692	–
leukemia	0.4425	mem	bug	0.17995	–	–
cod-rna	1.3650	2379.2676	1314.39	0.19312	0.30186	0.19464
w8a	4.9653	5240.28	2758.11	0.12206	0.18331	0.12296
skin-nonskin	0.7681	31.5212	21.9152	0.00089	0.00079	0.00079
ijcnn1	0.2667	941.2630	266.00	0.21391	0.21391	0.21391
madelon	36.0101	max iter	max time	0.51685	0.68631	0.66968
rcv1.binary	2.7404	max time	bug	0.19252	0.33324	–
gisette	13.7760	max time	max time	0.00013	0.31189	–
covtype.binary	0.3683	162.79	144.13	0.00000	0.00000	0.00000
real-sim	10.2459	max time	max time	0.14125	0.31605	–
news20	20.7303	mem	mem	0.32348	–	–

the point that $\beta(x_k)$ becomes zero, it is also the case that the value for $\|\phi(x_k)\|$ starts decreasing at a superlinear rate for a majority of the data sets (19/23), which is expected because of the superlinear convergence of $\{x_k\}$ as predicted by Theorem 35. For the other 4 datasets we can see that FaRSA converges on *covtype.binary* in a single iteration, and exhibits linear convergence for the data sets *skin-nonskin*, *w8a*, and *mushrooms*. This led us to use FaRSA on these three datasets with a tighter tolerance of 10^{-12} to better observe the asymptotic convergence; a superlinear rate of convergence of the iterates was then observed. A similar test was performed with the tolerance 10^{-12} on the 12 data sets for which $\beta(x_k)$ was nonzero at the final iterate in Table 5.9. The results

showed that eventually $\beta(x_k)$ was zero, again as predicted by Lemma 29(ii), for all the datasets that are successfully solved except *avazu-app.tr*, *kdda*, *kddb* and *url.combined*.

Table 5.9: The values of $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$ for the last 5 iterations.

Dataset	Measure	final-4	final-3	final-2	final-1	final
liver-disorder	$\ \beta(x_k)\ $	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	2.4e+00	2.3e-01	1.4e-02	2.5e-05	2.4e-11
fourclass	$\ \beta(x_k)\ $	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	1.5e+01	2.0e+00	1.2e-01	6.0e-04	1.8e-08
heart	$\ \beta(x_k)\ $	2.3e-02	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	2.2e-02	3.1e-02	5.8e-03	2.6e-04	5.4e-07
australian	$\ \beta(x_k)\ $	2.1e-03	1.7e-03	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	2.4e-03	1.7e-04	1.5e-03	1.2e-04	4.9e-07
diabetes	$\ \beta(x_k)\ $	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	2.9e-01	1.9e-02	1.0e-02	9.4e-05	1.8e-07
breast-cancer	$\ \beta(x_k)\ $	1.9e-01	5.5e-02	5.5e-02	1.4e-02	1.4e-02
	$\ \phi(x_k)\ $	9.0e-01	1.5e+02	2.6e-01	2.7e+01	8.0e-02
ionosphere	$\ \beta(x_k)\ $	2.0e-04	4.4e-04	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	2.4e-03	2.5e-04	1.6e-03	1.2e-04	1.4e-07
svmguide1	$\ \beta(x_k)\ $	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	5.6e-01	1.0e-01	1.1e-01	3.5e-03	2.7e-05
sonar	$\ \beta(x_k)\ $	7.2e-06	8.3e-04	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	1.1e-03	9.2e-03	6.9e-04	5.6e-05	6.3e-08
svmguide3	$\ \beta(x_k)\ $	9.9e-05	4.8e-05	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	6.2e-04	2.8e-05	8.5e-05	4.9e-06	6.7e-09
german.numer	$\ \beta(x_k)\ $	9.8e-04	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	7.6e-04	3.9e-03	5.4e-04	4.9e-05	2.2e-07
splice	$\ \beta(x_k)\ $	1.0e-03	1.1e-03	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	1.8e-03	1.3e-04	3.9e-04	2.6e-05	2.2e-09
mushrooms	$\ \beta(x_k)\ $	4.8e-06	0.0e+00	4.5e-06	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	4.5e-05	4.5e-05	9.5e-06	9.0e-06	3.6e-07

colon-cancer	$\ \beta(x_k)\ $	3.9e-04	6.0e-05	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	6.0e-05	8.4e-04	8.1e-05	7.6e-06	6.1e-09
a9a	$\ \beta(x_k)\ $	0.0e+00	5.6e-05	5.2e-06	0.0e+00	5.9e-08
	$\ \phi(x_k)\ $	7.8e-05	2.2e-04	1.1e-05	1.1e-06	1.0e-07
phishing	$\ \beta(x_k)\ $	2.3e-04	1.3e-04	1.9e-05	1.3e-06	1.6e-07
	$\ \phi(x_k)\ $	6.3e-04	3.8e-04	4.8e-05	4.4e-06	3.9e-07
duke-breast-cancer	$\ \beta(x_k)\ $	4.6e-04	4.2e-04	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	5.8e-04	1.6e-05	7.4e-04	3.6e-05	1.8e-07
leukemia	$\ \beta(x_k)\ $	5.3e-05	4.1e-05	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	5.6e-05	3.2e-06	1.8e-04	1.3e-05	3.2e-08
cod-rna	$\ \beta(x_k)\ $	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	9.2e-04	3.0e-03	5.7e-03	1.3e-03	4.9e-06
w8a	$\ \beta(x_k)\ $	3.7e-06	2.7e-06	1.7e-06	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	1.8e-05	1.1e-05	1.5e-06	4.1e-06	4.1e-07
skin-nonskin	$\ \beta(x_k)\ $	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	2.1e-03	8.7e-04	3.6e-04	1.5e-04	5.6e-05
ijcnn1	$\ \beta(x_k)\ $	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	4.5e-03	1.4e-03	1.3e-04	1.4e-06	1.5e-10
madelon	$\ \beta(x_k)\ $	6.1e-02	0.0e+00	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	6.9e-01	2.2e-02	2.1e-03	2.2e-05	1.4e-08
rcv1.binary	$\ \beta(x_k)\ $	1.5e-05	0.0e+00	1.9e-06	4.7e-07	5.3e-07
	$\ \phi(x_k)\ $	1.3e-05	1.9e-05	1.7e-05	2.1e-06	2.0e-07
gisette	$\ \beta(x_k)\ $	7.4e-03	7.8e-03	8.3e-03	1.3e-02	2.7e-05
	$\ \phi(x_k)\ $	1.1e-02	1.1e-02	1.2e-02	1.5e-02	1.1e-03
covtype.binary	$\ \beta(x_k)\ $	--	--	--	2.2e+03	0.0e+00
	$\ \phi(x_k)\ $	--	--	--	0.0e+00	5.4e-06
real-sim	$\ \beta(x_k)\ $	5.8e-06	5.5e-06	0.0e+00	2.0e-09	1.6e-08
	$\ \phi(x_k)\ $	6.8e-06	3.8e-06	6.5e-06	6.2e-06	6.0e-07
news20	$\ \beta(x_k)\ $	2.3e-05	1.4e-08	2.2e-07	1.8e-07	1.6e-07
	$\ \phi(x_k)\ $	6.4e-06	2.3e-05	2.2e-05	1.8e-06	1.5e-07

url_combined	$\ \beta(x_k)\ $	6.8e-07	1.1e-06	8.4e-07	1.3e-05	6.2e-07
	$\ \phi(x_k)\ $	1.8e-06	2.9e-06	1.9e-06	1.6e-05	1.5e-06
SUSY	$\ \beta(x_k)\ $	5.0e-05	3.5e-05	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	2.2e-04	6.8e-06	1.3e-04	7.7e-06	5.0e-08
avazu-app.tr	$\ \beta(x_k)\ $	1.0e-06	8.0e-08	2.4e-06	1.2e-06	8.6e-08
	$\ \phi(x_k)\ $	4.8e-07	1.0e-06	3.7e-06	4.6e-06	4.2e-07
kdda	$\ \beta(x_k)\ $	3.4e-05	2.6e-06	2.2e-06	1.7e-05	2.1e-06
	$\ \phi(x_k)\ $	4.7e-05	4.5e-06	3.3e-06	2.5e-05	2.4e-06
kddb	$\ \beta(x_k)\ $	2.4e-05	1.6e-04	1.4e-05	2.1e-05	5.4e-05
	$\ \phi(x_k)\ $	4.5e-05	2.7e-04	2.5e-05	4.2e-05	1.1e-04
HIGGS	$\ \beta(x_k)\ $	5.6e-04	5.6e-04	0.0e+00	0.0e+00	0.0e+00
	$\ \phi(x_k)\ $	6.2e-04	6.1e-05	1.1e-04	9.0e-06	6.9e-09
epsilon	$\ \beta(x_k)\ $	5.5e-06	5.1e-07	2.0e-06	3.4e-07	3.4e-07
	$\ \phi(x_k)\ $	5.2e-06	8.5e-06	1.1e-05	2.2e-06	2.2e-07

It is natural to consider the local convergence properties of the other solvers tested in Section 5.2. Obtaining empirical results is too costly to collect since the evolution of the accuracy measurements for these solver is not available on an iterate-wise basis. Instead, here we briefly summarize the theoretical local convergence properties of these solvers from the existing literature.

We start the summary for newGLMNET implemented in LIBLINEAR. Following [18, Appendix B], if $f(x)$ is strictly convex, then the objective function value converges at least linearly in newGLMNET. For ASACG [16], when the objective function is strongly convex, ASACG converges R-linearly to the global optimum provided the constraint multiplier is unique and a nondegeneracy condition holds. OBA is globalized by using the ISTA step as a reference for the desired progress. Although this enables OBA to achieve at least a linear rate-of-convergence for the iterates for strongly convex functions [17], empirically it often achieves superlinear convergence because it uses second-derivatives

in the subproblem. IRPN possesses a local superlinear convergence rate [73]. The algorithm FISTA (implemented in SPAMS) has a linear convergent rate for strongly convex objective functions. Finally, for the remaining methods (i.e. OWL-QN, Shooting, Gauss-Seidel, PSSas, and PSSgb), we are not aware of any local convergence results that have been established.

5.5.3 Testing accuracy for solutions produced by FaRSA

The previous sections have demonstrated the effectiveness of FaRSA in the sense of optimization metrics (e.g., computational time, robustness, number of required iterations, and local convergence rate). Now, we turn our attention to understanding the quality of the solutions returned by FaRSA in terms of the application itself, namely binary classification through logistic regression on the datasets from Table 5.2 and compare with other state-of-the-art solvers tested before. In particular, solving the target problem in Section 5.5.1 is referred to as training the model on the training datasets shown in Table 5.2. Once a model is trained (i.e., an approximate minimizer of parameters are obtained), the classification of new unseen data (called the testing set) is performed by using the computed approximate solution x to predict the labels for the test data. We remark here that not every training dataset in Table 5.2 has a corresponding testing dataset provided in the LIBSVM repository. Only 16 out of 35 datasets are provided testing datasets, which we present in Table 5.10. (We could create our own training and testing split for the other datasets, but in the spirit of reproducibility we prefer to simply use those provided in LIBSVM.) It is notable that the distribution of positive and negative testing instances are not uniform for all problems (e.g., the dataset *svmguide3* has 41 positive testing samples with no negative samples, whereas the dataset *duke-breast-cancer*, in contrast, only contains negative testing instances.

Table 5.10: Testing datasets for binary classification.

dataset	# of Samples	n	# of Positive	# of Negative
liver-disorders	200	6	100	100
svmguide1	4000	4	2000	2000
svmguide3	41	22	41	0
splice	2175	60	1131	1044
a9a	16281	123	3846	12435
duke-breast-cancer	4	7129	0	4
leukemia	34	7129	20	14
cod-rna	271617	8	90539	181078
w8a	14951	300	454	14497
ijcnn1	91701	22	8712	82989
madelon	600	500	300	300
rcv1.binary	677399	47236	355460	321939
avazu-app.tr	1953951	999990	252989	1700962
kdda	510302	20216830	442845	67457
kddb	748401	29890095	664374	84027
epsilon	100000	2000	49955	50045

We used solutions computed by FaRSA and other solvers under various termination tolerances ϵ between 10^{-1} and 10^{-6} to predict class labels on the testing datasets. We calculated the following four common evaluation measurements for binary classification: accuracy, precision, recall, and F1-score [93]:

$$\text{accuracy} := \frac{tp + tn}{tp + tn + fp + fn} \quad (5.10a)$$

$$\text{precision} := \frac{tp}{tp + fp} \quad (5.10b)$$

$$\text{recall} := \frac{tp}{tp + fn} \quad (5.10c)$$

$$\text{F1-score} := \frac{2\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (5.10d)$$

where tp is the number of true positive (positive instances predicted as positive), tn the number of true negative (negative instances predicted as negative), fp the number of false positives (negative instances predicted as positive), and fn the number of false negatives (positive instances predicted as negative).

The testing results are presented in Figures 5.2–5.9. Figure 5.2 and Figure 5.3 show the accuracies (see (5.10a)), Figure 5.4 and Figure 5.5 depict the precisions (see (5.10b)), the recalls (5.10c) are described in Figure 5.6 and Figure 5.7, and the F1-scores (5.10d) are plotted in Figure 5.8 and Figure 5.9 for these solvers on the data sets tested.

We may see in these figures that the curves are similar in the high accuracy regime (i.e., when a small tolerance is used). This indicates that when high accuracy solutions are produced by all solvers, they possess similar prediction performance on unseen data. More specifically, FaRSA achieves high precision, recall, accuracy, and F1 score on *rcv1.binary*, *leukemia*, *cod-rna*, *kdda*, *kddb*, and *epsilon*; high accuracy on *ijcnn1*, *w8a*, *a9a*, and *avazu-app.tr*; high precision on *splice_scale* and *svmguide3*; and high recall on *svmguide1*.

For the majority of the testing datasets (e.g., *leukemia*, *cod-rna*, *w8a*, and *ijcnn1*), we can observe that these evaluation metrics increase as the tolerance decreases, i.e., more accurate solutions on the training datasets tends to produce solutions that obtain smaller prediction error on the testing datasets. However, overfitting does appear on some of the remaining testing sets whose evaluation measurements decrease a little bit following tighter tolerances (e.g., see *kdda* and *kddb*). There are also a few testing datasets for which the solutions of FaRSA do not achieve good performance (e.g., *liver-disorders* and *madelon*). The reason for bad performance on these problems is not clear, but appears to be related to insufficient quality in the training datasets for learning a reliable model. For *duke-breast-cancer*, only accuracy and precision are plotted because the recall and F1-score are invalid for testing sets with no positive instances. It is also interesting that FaRSA produces models of high quality on the three big datasets *kdda*, *kddb*, and *epsilon*. For the other remaining big dataset *avazu-app.tr*, since its number of positive and negative data instances

are significantly imbalanced, FaRSA only reaches high accuracy.

In summary, we may conclude that solutions produced by FaRSA on these commonly used testing datasets (data not used to train the model) generally produce quality predictive models for ℓ_1 -regularized logistic regression. In particular, the solutions produced by FaRSA perform competitively when compared to other state-of-the-art solvers for predicting labels of unseen data.

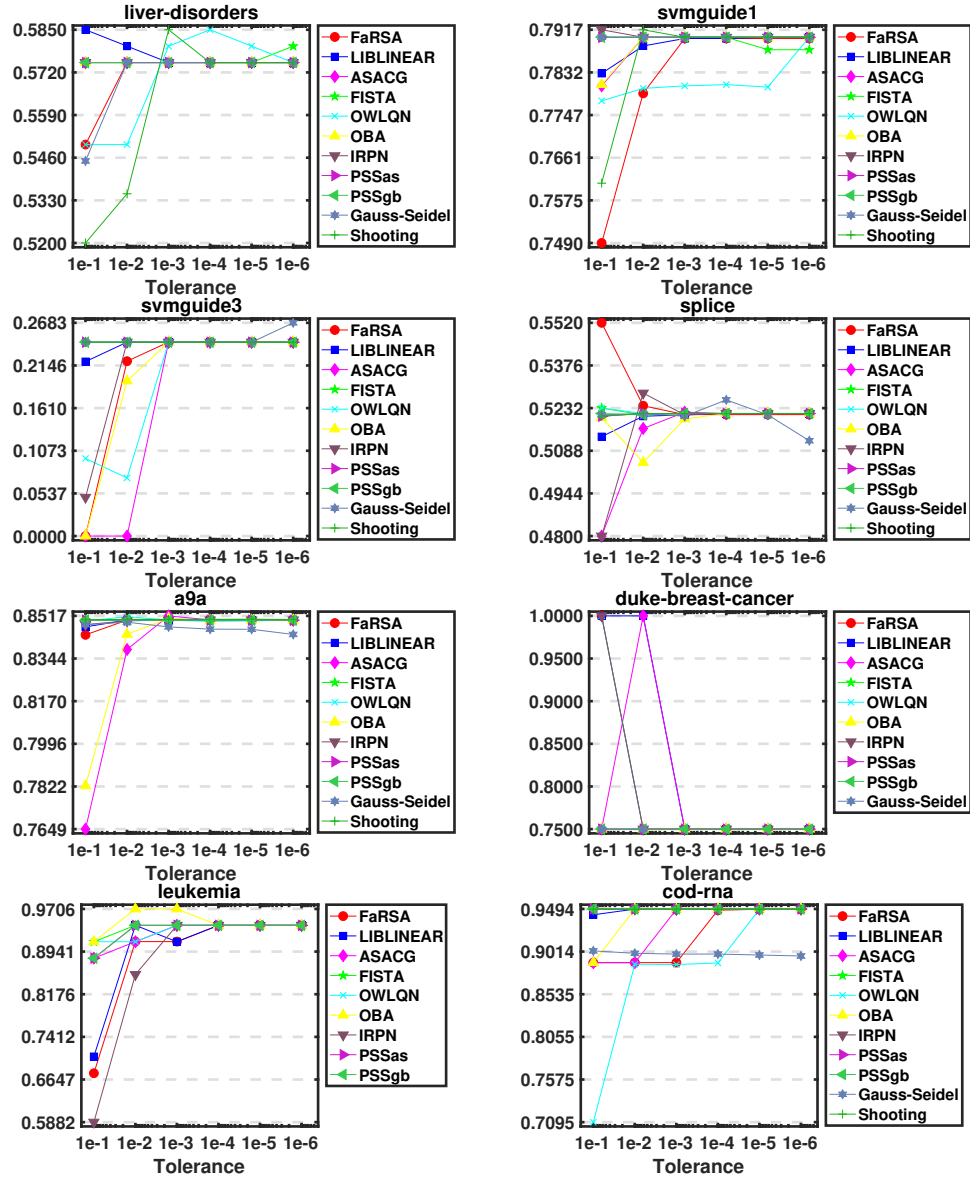


Figure 5.2: Accuracy (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).

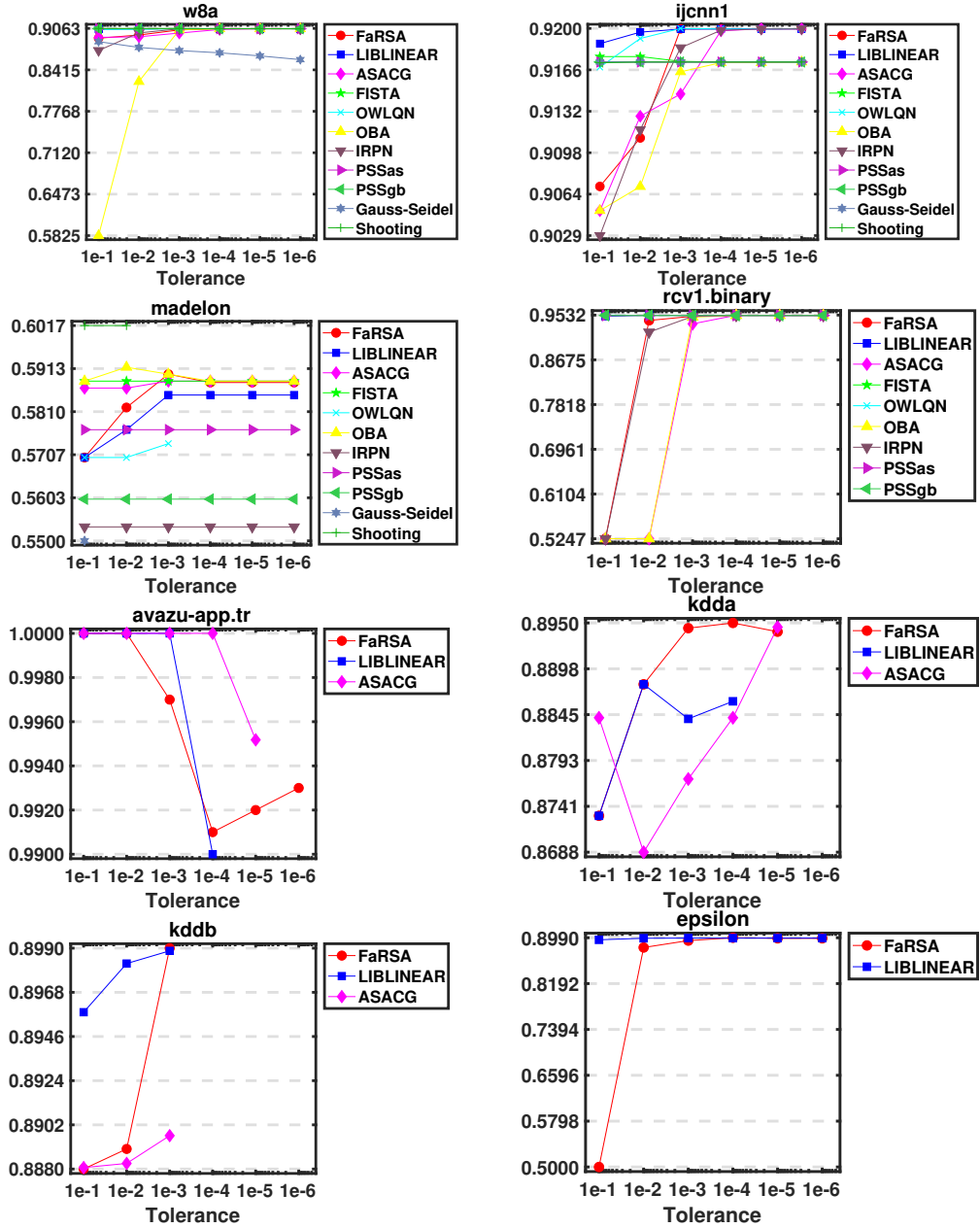


Figure 5.3: Accuracy (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).

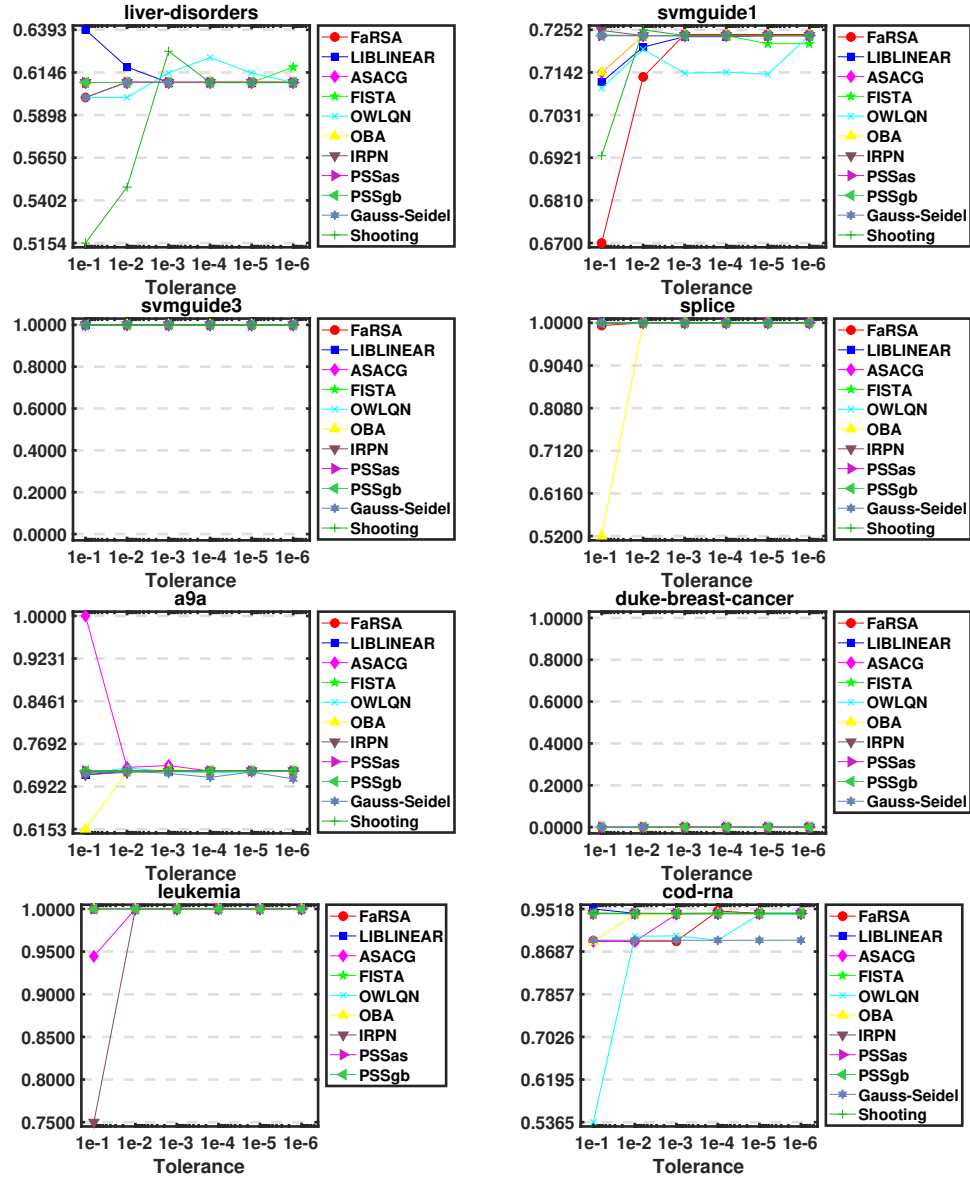


Figure 5.4: Precision (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).

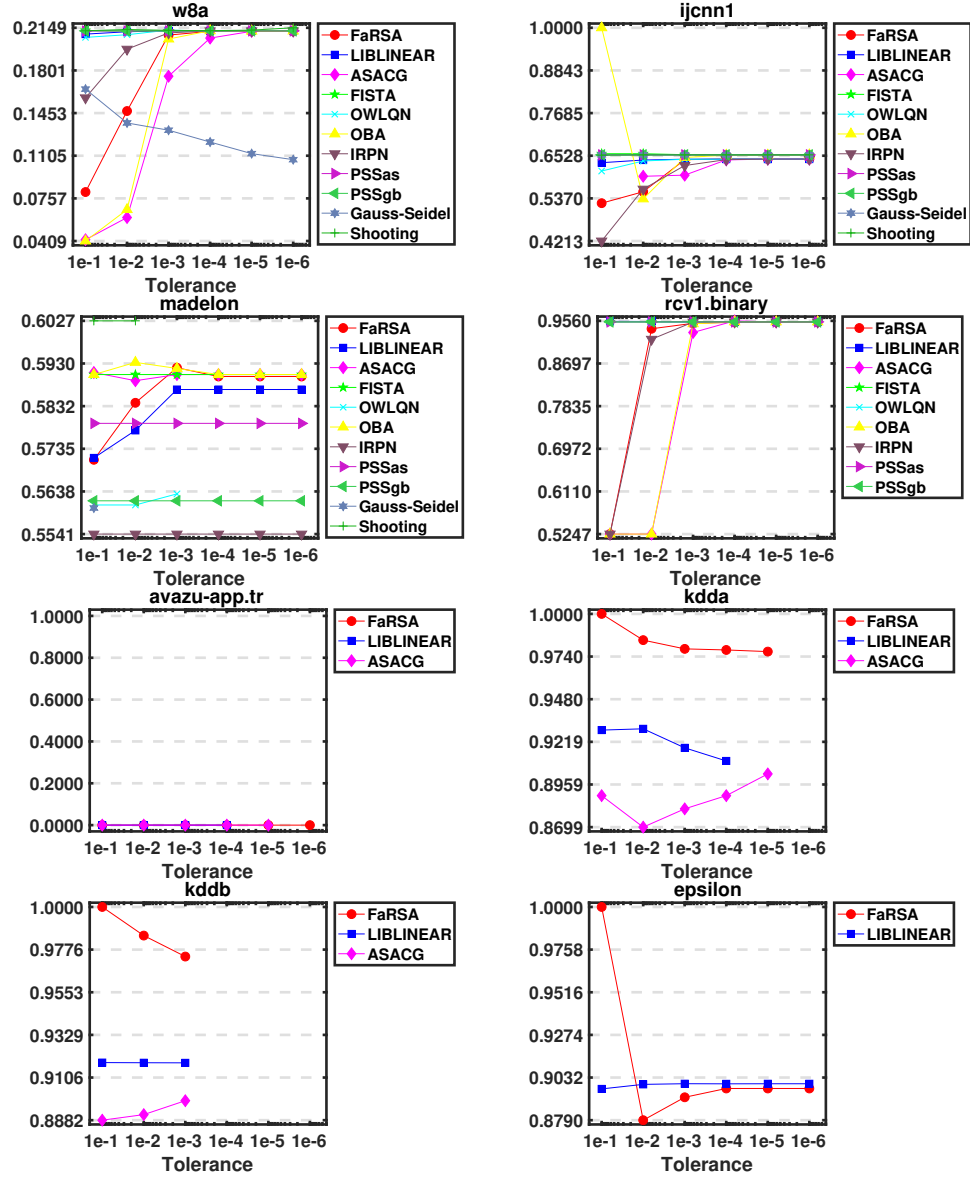


Figure 5.5: Precision (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).

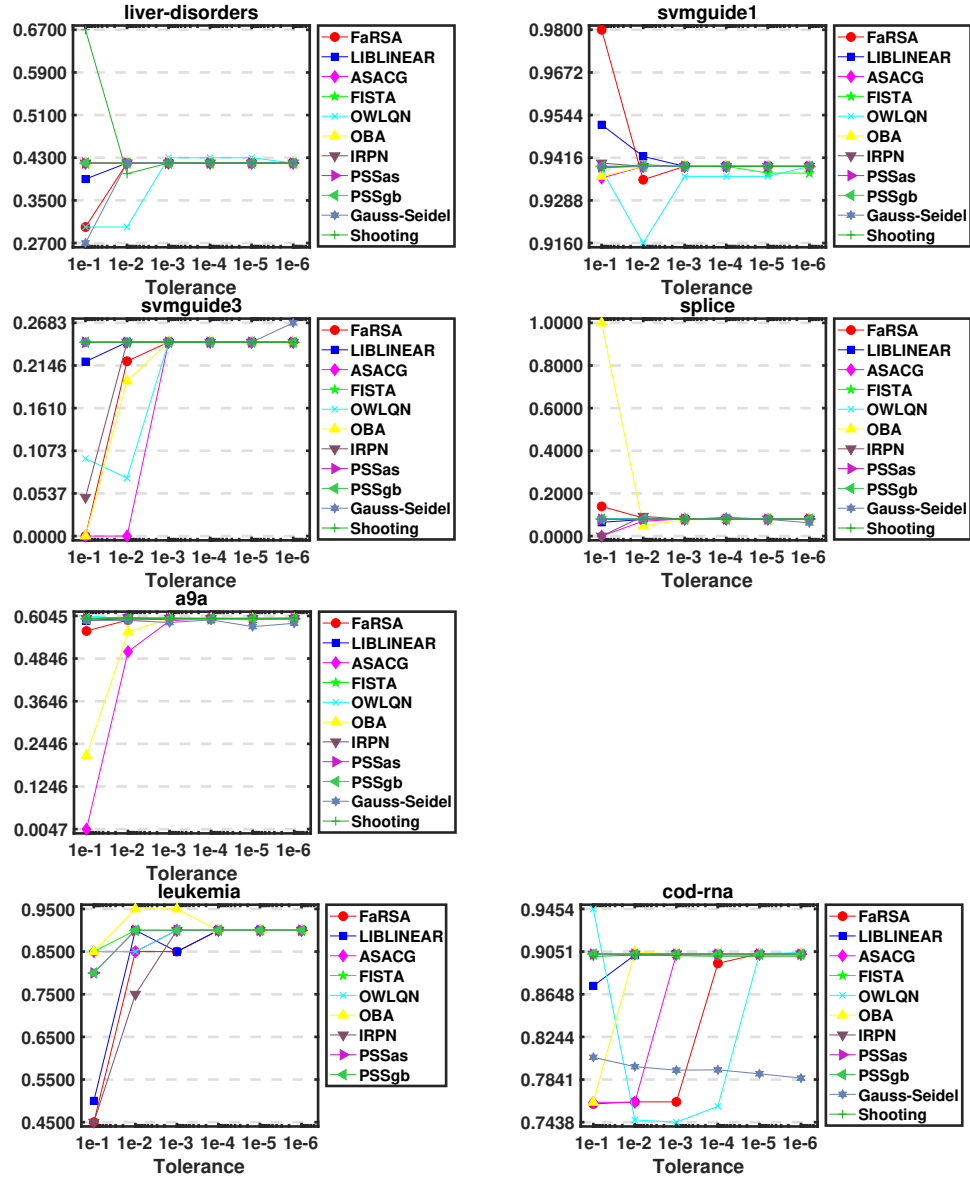


Figure 5.6: Recall (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).

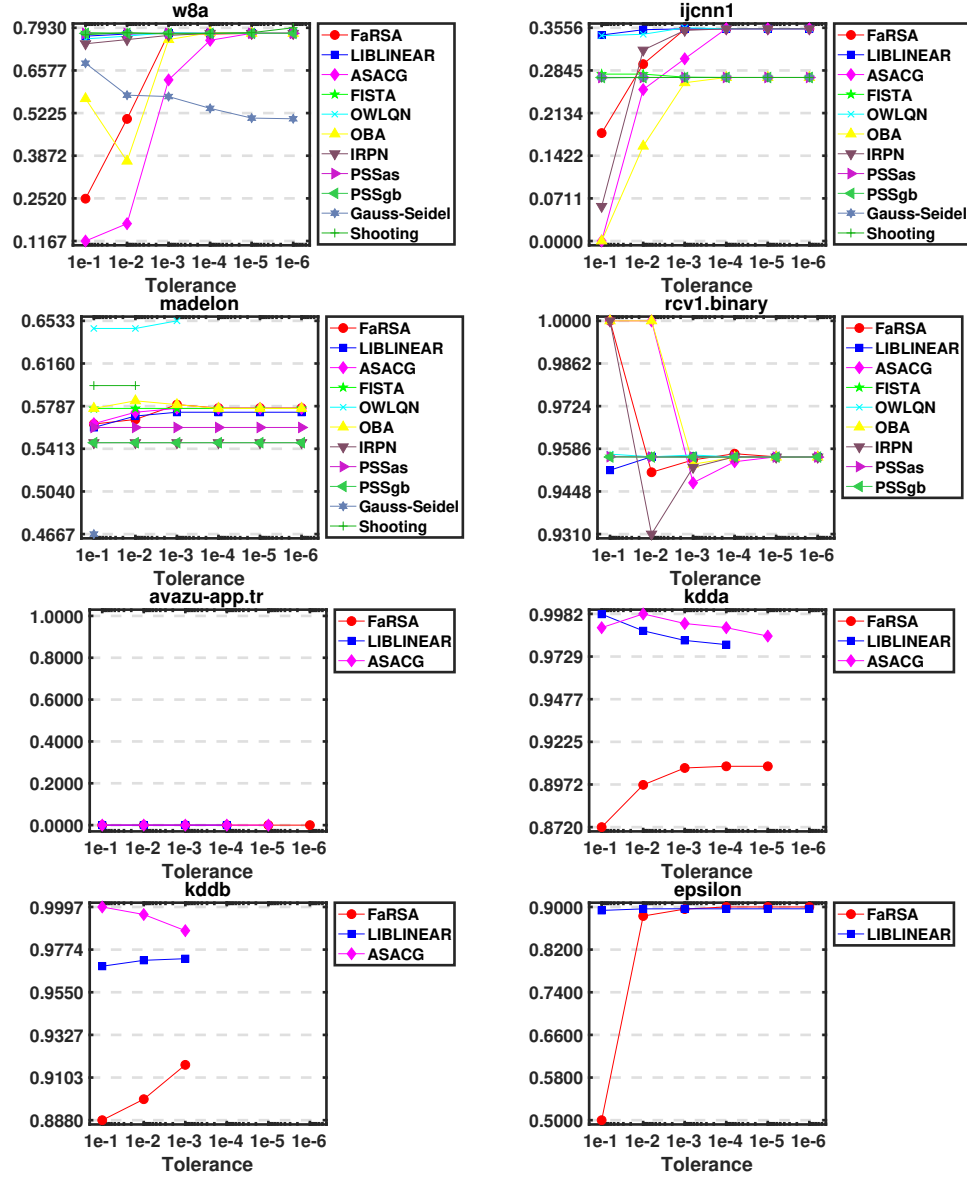


Figure 5.7: Recall (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).

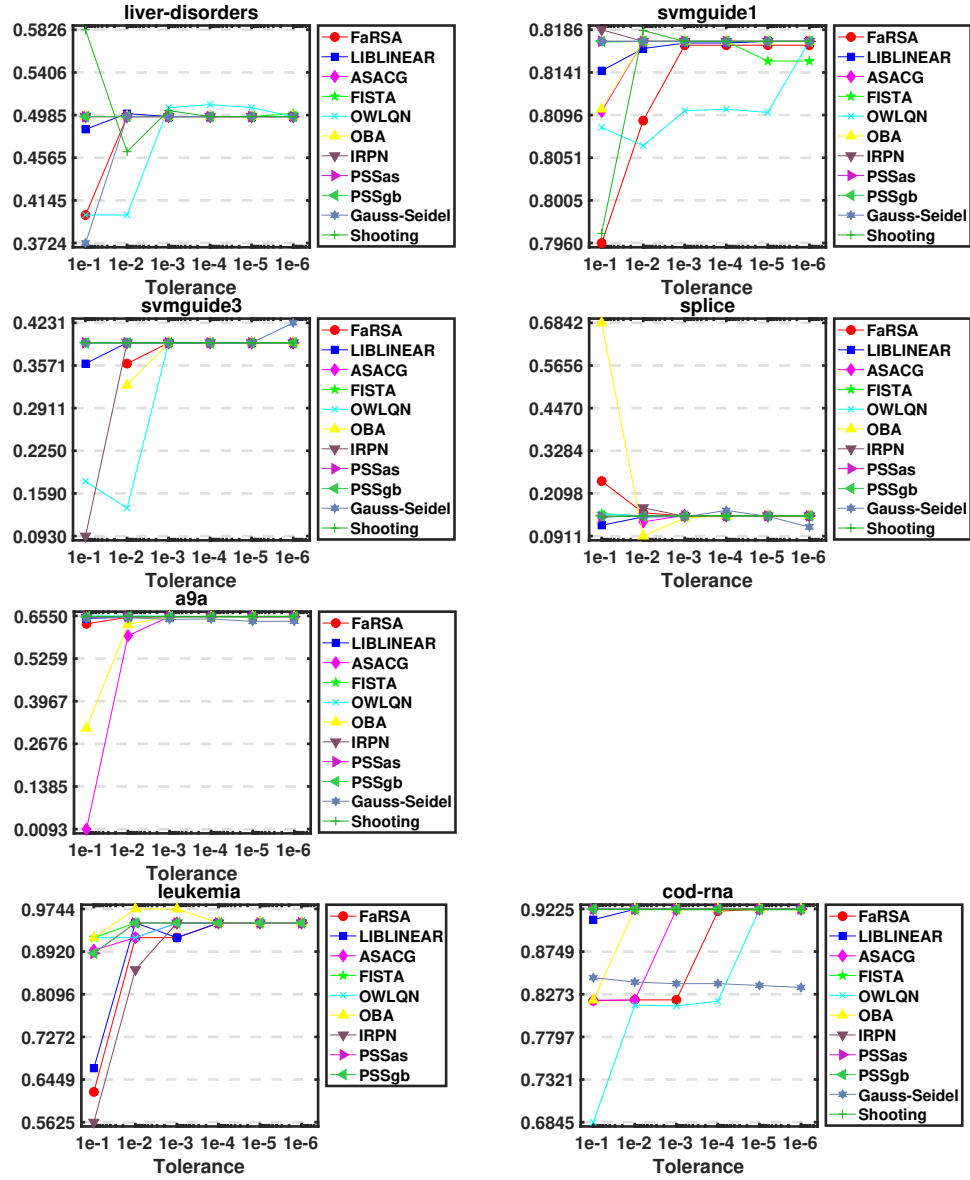


Figure 5.8: F1-score (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).

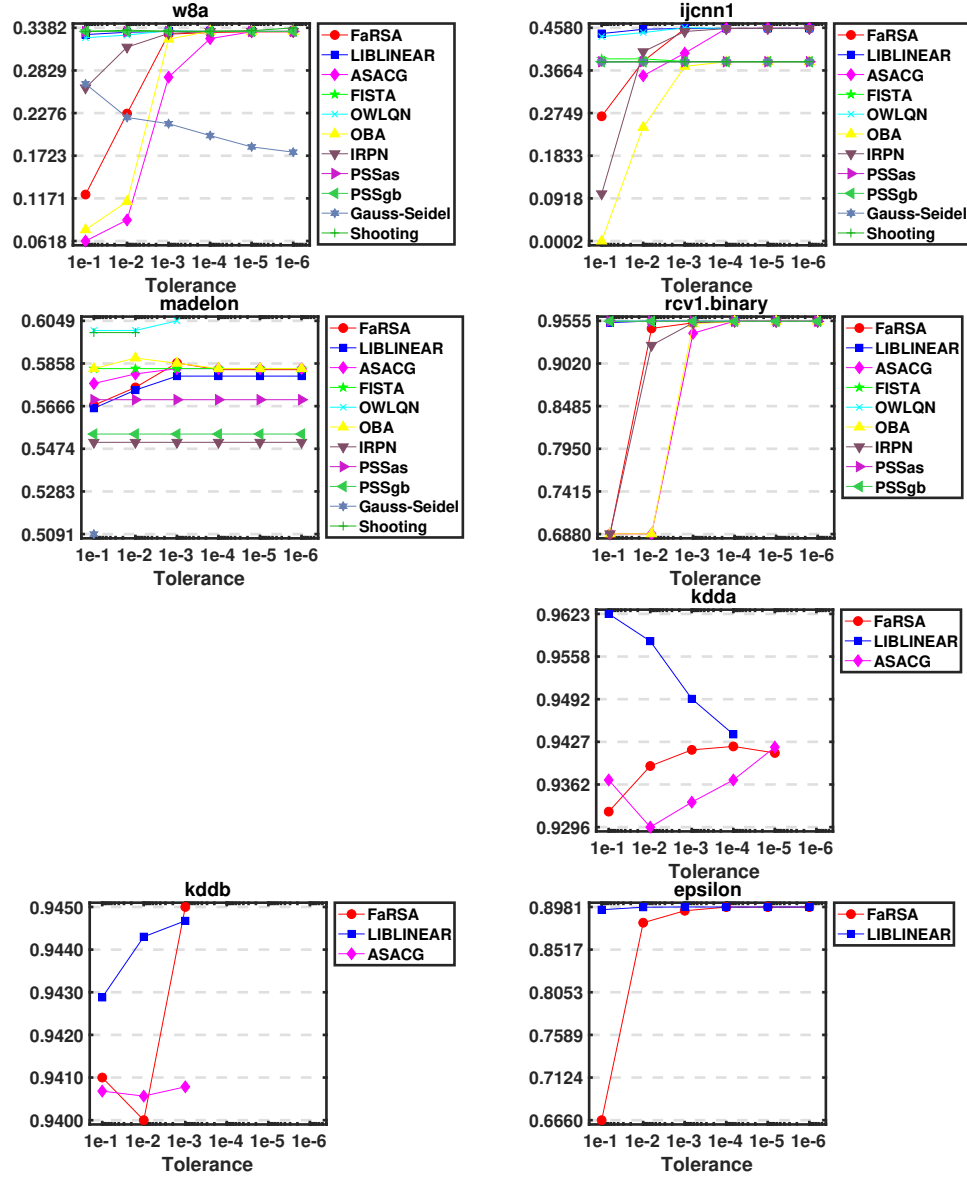


Figure 5.9: F1-score (vertical-axis) of solvers on predicting labels for testing datasets over different tolerance levels (horizontal-axis).

Chapter 6

Extensions

Regarding the ℓ_1 -norm regularizer, the zero/non-zero structure of each component of a solution to the optimization problem (1.1) is individually determined by the chosen weight λ ; in particular, any existing relationships and structures between the variables (e.g., spatial, hierarchical or physics of the problem at hand) are merely disregarded [9, 10]. However, in many practical situations the solutions obtained from the optimization problem may benefit by including prior knowledge aimed at improving interpretability predictive performance. This real-world requirement motivates the need to design structured sparsity-inducing regularization schemes that are capable of encoding more sophisticated prior knowledge about the expected sparsity patterns. The most natural form of structured sparsity is arguably group sparsity, which is based on using prior knowledge that pre-specified disjoint blocks of variables should be selected (nonzero variables) or ignored (zero variables) simultaneously [94, 95, 96]. There also exist regularizers based on overlapping groups of variables that represent more sophisticated hierarchical sparsity structure [9, 97, 98, 99, 100].

In this chapter, we discuss an extension of FaRSA for solving the structured

sparsity-inducing regularization problem (6.1):

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) + \lambda \Omega(x) \quad (6.1)$$

where we assume that f is a convex and twice continuously differentiable function, $\lambda > 0$, and Ω is a structured sparsity-inducing function. In particular, we extend FaRSA to disjoint group structured sparsity-inducing problems.

In this chapter, we introduce the ℓ_1/ℓ_p -regularization problem along with its useful properties in Section 6.1. Next, the proximal operator for the ℓ_1/ℓ_p -regularization problem is discussed in Section 6.2. We then present one extended algorithmic framework of FaRSA in Section 6.3 for such regularizations.

6.1 The ℓ_1/ℓ_p -Regularization Problem

A natural form of structured sparsity is disjoint group sparsity, which aims to ensure that disjoint blocks of variables are selected (nonzero) or ignored (zero) simultaneously. Similar to ℓ_1 -regularization, group sparsity can be achieved by augmenting the objective function with specifically designed sparsity inducing norms. Before introducing the desired sparsity inducing norm, we first introduce the concept of a mixed $\ell_{q,p}$ -norm over $\mathbb{R}^{n \times m}$ in the following definition.

Definition 36 (Mixed $\ell_{q,p}$ -norm). *The mixed $\ell_{q,p}$ -norm is defined as*

$$\|A\|_{q,p} = \left(\sum_{j=1}^m \left(\sum_{i=1}^n w_{ij} |a_{ij}|^p \right)^{q/p} \right)^{1/q} \quad (6.2)$$

for any $A \in \mathbb{R}^{n \times m}$ and $w_{ij} > 0$ for all $i \in \{1, 2, \dots, n\}$ and $j \in \{1, 2, \dots, m\}$.

Now for disjoint group sparsity, we use \mathcal{G} to denote a collection of subsets of variables (groups) such that they form a partition of $\mathcal{I} = \{1, 2, \dots, n\}$, and let

w_g denote a positive scalar weight for each $g \in \mathcal{G}$. A popular sparsity inducing norm is then defined as follows (it may be viewed as a special case of the mixed $\ell_{q,p}$ -norm with $m = 1$ and $q = 1$, but that also accounts for the group partition).

Definition 37 (Mixed ℓ_1/ℓ_p -norm). *For $x \in \mathbb{R}^n$, define*

$$\Omega(x) = \sum_{g \in \mathcal{G}} w_g \|x_g\|_p = \sum_{g \in \mathcal{G}} w_g \left(\sum_{i \in g} |x_i|^p \right)^{1/p} \text{ for any } p \in [1, \infty] \quad (6.3)$$

where \mathcal{G} is a partition of $\mathcal{I} = \{1, 2, \dots, n\}$ (i.e. $\bigcup_{g \in \mathcal{G}} g = \mathcal{I}$ and $g_i \cap g_j = \emptyset$ for any $g_i \neq g_j \in \mathcal{G}$), w_g is a positive weight, and $x_g \in \mathbb{R}^{|g|}$ for each $g \in \mathcal{G}$. This norm is usually referred to as a mixed ℓ_1/ℓ_p -norm.

If $0 < p < 1$, then Ω as defined in (6.3) is not a norm when at least one group contains at least two elements, as we now show.

Lemma 38. *If $0 < p < 1$, \mathcal{G} is a partition of $\mathcal{I} = \{1, 2, \dots, n\}$, $w_g > 0$ for all $g \in \mathcal{G}$, and there exists at least one $g' \in \mathcal{G}$ such that $|g'| \geq 2$, then Ω as defined in (6.3) is not a norm.*

Proof. Let $e_i \in \mathbb{R}^n$ be a vector with i th entry equal to one ($i \in g'$) and other entries equal to zero. Let $e_j \in \mathbb{R}^n$ be a vector with j th entry equal to one ($j \in g'$ and $i \neq j$) and other entries equal to zero. Since $p \in (0, 1)$, it follows that

$$\begin{aligned} \Omega(e_i + e_j) &= \sum_{g \in \mathcal{G}} w_g \|[e_i + e_j]_g\|_p = w_{g'} \|[e_i + e_j]_{g'}\|_p = w_{g'} \cdot 2^{\frac{1}{p}} \\ &> w_{g'} \cdot 2 = w_{g'} (\|e_i\|_p + \|e_j\|_p) \\ &= \sum_{g \in \mathcal{G}} w_g \|[e_i]_g\|_p + \sum_{g \in \mathcal{G}} w_g \|[e_j]_g\|_p \\ &= \Omega(e_i) + \Omega(e_j), \end{aligned} \quad (6.4)$$

which means that Ω does not satisfy the triangle property. Consequently, we must conclude that Ω is not a norm, which completes the proof. \square

For $p \geq 1$, we may show that the mixed ℓ_1/ℓ_p -norm is a convex function.

Lemma 39. *If $\Omega(x) = \sum_{g \in \mathcal{G}} w_g \|x_g\|_p$ is a mixed ℓ_1/ℓ_p -norm as given by Definition 37, then Ω is a convex function over \mathbb{R}^n .*

Proof. For any $\mu \in [0, 1]$ and any $\{x_1, x_2\} \subset \mathbb{R}^n$, it follows from the triangle inequality and absolute summability that

$$\begin{aligned}
\Omega(\mu x_1 + (1 - \mu)x_2) &= \sum_{g \in \mathcal{G}} w_g \|[\mu x_1 + (1 - \mu)x_2]_g\|_p \\
&\leq \sum_{g \in \mathcal{G}} w_g [\mu \|x_1\|_p + (1 - \mu) \|x_2\|_p] \\
&= \mu \sum_{g \in \mathcal{G}} w_g \|x_1\|_p + (1 - \mu) \sum_{g \in \mathcal{G}} w_g \|x_2\|_p \\
&= \mu \Omega(x_1) + (1 - \mu) \Omega(x_2).
\end{aligned} \tag{6.5}$$

Thus, it follows that Ω is a convex function over \mathbb{R}^n . \square

In this thesis, we skip the case $p < 1$ since the corresponding Ω is usually not a norm (see Lemma 38), and can easily be shown to usually be nonconvex as well. Thus, the ℓ_1/ℓ_p -regularization problem that we consider is given by

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \left\{ f(x) + \lambda \Omega(x) = f(x) + \lambda \sum_{g \in \mathcal{G}} w_g \|x_g\|_p =: F(x) \right\} \tag{6.6}$$

where f is a convex and twice continuously differentiable function, \mathcal{G} is a partition of $\{1, 2, \dots, n\}$, and $w_g > 0$ for all $g \in \mathcal{G}$. Popular choices are $p \in \{2, \infty\}$. Note that if $p = 2$, $\mathcal{G} = \{\{1\}, \{2\}, \dots, \{n\}\}$, and $w_g = 1$ for all $g \in \mathcal{G}$, then $\Omega(x) = \|x\|_1$, i.e., the ℓ_1 -norm is a particular instance of the mixed ℓ_1/ℓ_p -norm. In general, using the mixed ℓ_1/ℓ_p -norm produces solutions such that variables in the same group are nonzero or zero simultaneously.

Similar to when ℓ_1 -regularization is used in optimization, proximal methods are popular for solving ℓ_1/ℓ_p -regularization problems. We will discuss the prox-

imal operator for the mixed ℓ_1/ℓ_p -norm in the next section, which is crucial for extending FaRSA to the mixed-norm setting. Before that discussion, we need to introduce a few more concepts. First, we introduce the dual norm, which is important in the study of sparsity-inducing regularization [10, 101].

Definition 40 (Dual norm). *The dual norm v^* for norm $v : \mathbb{R}^n \rightarrow [0, +\infty)$ is*

$$v^*(x) := \sup_{z \in \mathbb{R}^n} x^T z \text{ subject to } v(z) \leq 1. \quad (6.7)$$

It is well known that the dual of the dual norm is the original norm, and that for the ℓ_p -norm ($p \in [1, \infty]$) the dual norm is the $\ell_{p'}$ -norm with $p' \in [1, \infty]$ satisfying $\frac{1}{p} + \frac{1}{p'} = 1$ (using the convention that $1/\infty = 0$). Thus, the dual norm of the ℓ_1 -norm is the ℓ_∞ -norm, and the ℓ_2 -norm is self-dual. On the other hand, for the mixed ℓ_1/ℓ_p -norm, the dual norm is the mixed $\ell_\infty/\ell_{p'}$ -norm where $\frac{1}{p} + \frac{1}{p'} = 1$, which is proved in the following lemma.

Lemma 41. *For the mixed ℓ_1/ℓ_p -norm in Definition 37, its dual norm is the mixed $\ell_\infty/\ell_{p'}$ -norm given by*

$$\Omega^*(x) = \max_{g \in \mathcal{G}} \left\{ \frac{1}{w_g} \|x_g\|_{p'} \right\} \quad (6.8)$$

where $\frac{1}{p} + \frac{1}{p'} = 1$.

Proof. By the definition of dual norm, $\Omega^*(x)$ is

$$\Omega^*(x) = \sup_{z \in \mathbb{R}^n} z^T x \text{ subject to } \Omega(z) \leq 1.$$

For any feasible $z \in \mathbb{R}^n$, it follows that

$$\Omega(z) = \sum_{g \in \mathcal{G}} w_g \|z_g\|_p \leq 1. \quad (6.9)$$

Recalling that $p \geq 1$, it follows from $w_g > 0$ for all $g \in \mathcal{G}$, (6.9), $1/p + 1/p' = 1$, Hölder's inequality [102], and the fact that \mathcal{G} forms a disjoint partition that

$$\begin{aligned} z^T x &= \sum_{g \in \mathcal{G}} z_g^T x_g \leq \sum_{g \in \mathcal{G}} |z_g^T x_g| \leq \sum_{g \in \mathcal{G}} \|z_g\|_p \|x_g\|_{p'} = \sum_{g \in \mathcal{G}} w_g \|z_g\|_p \frac{1}{w_g} \|x_g\|_{p'} \\ &\leq \max_{g \in \mathcal{G}} \left\{ \frac{1}{w_g} \|x_g\|_{p'} \right\} \cdot \sum_{g \in \mathcal{G}} w_g \|z_g\|_p \leq \max_{g \in \mathcal{G}} \left\{ \frac{1}{w_g} \|x_g\|_{p'} \right\} \end{aligned} \quad (6.10)$$

for all $x \in \mathbb{R}^n$ and all feasible $z \in \mathbb{R}^n$. Thus, $\Omega^*(x)$ is upper-bounded by $\max_{g \in \mathcal{G}} \{\|x_g\|_{p'}/w_g\}$. To complete the proof, it suffices to find a feasible \hat{z} that obtains this upper-bound, which we proceed to do now.

Let us define $g^* = \operatorname{argmax}_{g \in \mathcal{G}} \{\|x_g\|_{p'}/w_g\}$ so that

$$\frac{1}{w_{g^*}} \|x_{g^*}\|_{p'} = \max_{g \in \mathcal{G}} \left\{ \frac{1}{w_g} \|x_g\|_{p'} \right\}. \quad (6.11)$$

Now, let us first construct $\bar{z} \in \mathbb{R}^n$ such that $\bar{z}_i := \operatorname{sign}(x_i)|x_i|^{p'-1}$ for all $i \in g^*$, and $\bar{z}_i := 0$ for all $i \notin g^*$. We can then compute that

$$\bar{z}^T x = \sum_{g \in \mathcal{G}} \bar{z}_g^T x_g = \bar{z}_{g^*}^T x_{g^*} = \sum_{i \in g^*} x_i \operatorname{sign}(x_i) |x_i|^{p'-1} = \sum_{i \in g^*} |x_i|^{p'} = \|x_{g^*}\|_{p'}^{p'}. \quad (6.12)$$

We can further calculate that

$$\begin{aligned} \|\bar{z}\|_p^p &= \sum_{g \in \mathcal{G}} \|\bar{z}_g\|_p^p = \sum_{i \in g^*} |\bar{z}_i|^p = \sum_{i \in g^*} |\operatorname{sign}(x_i)|x_i|^{p'-1}|^p \\ &= \sum_{i \in g^*} |x_i|^{p(p'-1)} = \sum_{i \in g^*} |x_i|^{p'} = \|x_{g^*}\|_{p'}^{p'} \end{aligned} \quad (6.13)$$

where we used the fact that $p(p' - 1) = p'$ since $1/p + 1/p' = 1$.

Now, using \bar{z} , we construct $\hat{z} \in \mathbb{R}^n$ such that $\hat{z}_{g^*} = \frac{\bar{z}_{g^*}}{w_{g^*} \|\bar{z}_{g^*}\|_p}$ and the remaining

entries are set to zero. It is easy to verify that \hat{z} is feasible since

$$\Omega(\hat{z}) = \sum_{g \in \mathcal{G}} w_g \|\hat{z}_g\|_p = w_{g^*} \frac{\|\bar{z}_{g^*}\|_p}{w_{g^*} \|\bar{z}_{g^*}\|_p} = 1.$$

It follows from the construction of \hat{z} , (6.11), (6.12), and (6.13) that

$$\begin{aligned} \hat{z}^T x &= \sum_{g \in \mathcal{G}} \hat{z}_g^T x_g = \hat{z}_{g^*}^T x_{g^*} = \frac{\bar{z}_{g^*}^T x_{g^*}}{w_{g^*} \|\bar{z}_{g^*}\|_p} = \frac{\|x_{g^*}\|_{p'}^{p'}}{w_{g^*} \|x_{g^*}\|_{p'}^{p'/p}} \\ &= \frac{1}{w_{g^*}} \|x_{g^*}\|_{p'}^{p' - p'/p} = \frac{1}{w_{g^*}} \|x_{g^*}\|_{p'} = \max_{g \in \mathcal{G}} \left\{ \frac{1}{w_g} \|x_g\|_{p'} \right\}, \end{aligned} \quad (6.14)$$

where we also used that $p' - p'/p = 1$ since $1/p + 1/p' = 1$. Since the feasible point \hat{z} is such that $\hat{z}^T x$ achieves the upper bound on $z^T x$ as given by (6.10), we must conclude that $\Omega^*(x) = \max_{g \in \mathcal{G}} \|x_g\|_{p'}/w_g$, as claimed. \square

The next concept to be introduced is the standard orthogonal projection operator. We note that this projection operator is different from the *orthantwise* projection operator used earlier (see Definition 11).

Definition 42 (Projection Operator onto Convex Set (Orthogonal Projection)).

Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a nonempty closed convex set. We define the projection of a given vector $y \in \mathbb{R}^n$ onto \mathcal{X} as

$$\text{Proj}_{\mathcal{X}}(y) = \underset{x \in \mathcal{X}}{\text{argmin}} \|x - y\|_2.$$

We are now able to discuss the proximal operator for the mixed ℓ_1/ℓ_p -norm.

6.2 Proximal Operator for the ℓ_1/ℓ_p -Norm

Let us recall the definition of the proximal operator. Given a function $h : \mathbb{R}^n \rightarrow \mathbb{R}$ and a scalar $\lambda > 0$, the proximal operator for λh maps the vector $\mu \in \mathbb{R}^n$ to

$$\text{Prox}_{\lambda h}(\mu) = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2} \|x - \mu\|^2 + \lambda h(x). \quad (6.15)$$

To compute $\text{Prox}_{\lambda h}(\mu)$ efficiently, we may utilize a relationship between $\text{Prox}_{\lambda h}$ and the dual norm of h , which shows that the proximal operator can always be computed as the residual of a Euclidean projection onto a certain convex set.

Lemma 43 (Relation to dual norm). *Let $\text{Prox}_{\lambda h}$ be the proximal operator associated with the regularization $\lambda h(x)$ for some norm $h(x)$. It follows that*

$$\text{Prox}_{\lambda h} = I_n - \text{Proj}_{\mathcal{B}(h^*, \lambda)} \quad (6.16)$$

where $\mathcal{B}(h^*, \lambda) := \{x \in \mathbb{R}^n : h^*(x) \leq \lambda\}$. Consequently, for all $\mu \in \mathbb{R}^n$,

$$\text{Prox}_{\lambda h}(\mu) = \mu - \text{Proj}_{\mathcal{B}(h^*, \lambda)}(\mu). \quad (6.17)$$

Proof. See the proof of Lemma 1.3 in [103]. □

Lemma 43 is quite useful since computing $\text{Prox}_{\lambda h}(\mu)$ is transformed into computing a projection onto the ball of radius λ of the dual norm function h^* . In particular, it follows from Lemma 43 for $h(x) = \Omega(x)$ that

$$\text{Prox}_{\lambda \Omega}(\mu) = \mu - \text{Proj}_{\mathcal{B}(\Omega^*, \lambda)}(\mu). \quad (6.18)$$

Hence, to calculate $\text{Prox}_{\lambda \Omega}(\cdot)$ efficiently, we may use Ω^* as given in Lemma 41.

We now present a separable sum property for proximal operators from [104,

Chapter 2.1], which will be another useful property used to extend FaRSA.

Lemma 44. *Let \mathcal{G} be a partition of $\{1, 2, \dots, n\}$ and $h : \mathbb{R}^n \rightarrow \mathbb{R}$ be a separable function with respect to \mathcal{G} (i.e. $h(x) = \sum_{g \in \mathcal{G}} h_g(x_g)$ for some functions $\{h_g\}$ and for all $x \in \mathbb{R}^n$), then*

$$[\text{Prox}_h(\mu)]_g = \text{Prox}_{h_g}(\mu_g) \quad (6.19)$$

holds for all $\mu \in \mathbb{R}^n$ and $g \in \mathcal{G}$.

The next corollary further shows how to decompose the calculation of $\text{Proj}_{\mathcal{B}(\Omega^*, \lambda)}(\mu)$ into a sequence of projections over reduced-spaces.

Corollary 45. *If \mathcal{G} and h are as in Lemma 44, then*

$$[\text{Proj}_{\mathcal{B}(h^*, \lambda)}(\mu)]_g = \text{Proj}_{\mathcal{B}(h_g^*, \lambda)}(\mu_g) \quad (6.20)$$

holds for all $\mu \in \mathbb{R}^n$ and $g \in \mathcal{G}$.

Proof. For all $\mu \in \mathbb{R}^n$ and $g \in \mathcal{G}$, it follows Lemma 43 and Lemma 44 that

$$\begin{aligned} [\text{Prox}_{\lambda h}(\mu)]_g &= \mu_g - [\text{Proj}_{\mathcal{B}(h^*, \lambda)}(\mu)]_g \quad \text{and} \\ [\text{Prox}_{\lambda h}(\mu)]_g &= \text{Prox}_{\lambda h_g}(\mu_g) = \mu_g - \text{Proj}_{\mathcal{B}(h_g^*, \lambda)}(\mu_g). \end{aligned}$$

The identity (6.20) follows by combining the two previous equalities. \square

Using Lemma 41 and Corollary 45 allows us to compute $\text{Proj}_{\mathcal{B}(\Omega^*, \lambda)}(\mu)$ by computing some orthogonal projections, each of which involves a subset of the variables defined via the partition \mathcal{G} . Specifically, one obtains that

$$[\text{Proj}_{\mathcal{B}(\Omega^*, \lambda)}(\mu)]_g = \text{Proj}_{\mathcal{B}(\|\cdot\|_{p'}, \lambda w_g)}(\mu_g) \quad (6.21)$$

for all $g \in \mathcal{G}$. Let us consider examples for two choices of p .

- Let $p = 2$ so that Ω is the mixed ℓ_1/ℓ_2 -norm. In this case, it follows from Lemma 41 and (6.21) that

$$[\text{Proj}_{\mathcal{B}(\Omega^*, \lambda)}(\mu)]_g = \text{Proj}_{\mathcal{B}(\|\cdot\|_2, \lambda w_g)}(\mu_g) = \min\left(1, \frac{\lambda w_g}{\|\mu_g\|}\right) \mu_g \quad (6.22)$$

for each $g \in \mathcal{G}$. Furthermore, by now combining (6.22) and Lemma 43 we can compute the proximal operator for the mixed ℓ_1/ℓ_2 -norm as

$$[\text{Prox}_{\lambda\Omega}(\mu)]_g = \max\left\{0, 1 - \frac{\lambda w_g}{\|\mu_g\|}\right\} \cdot \mu_g \quad (6.23)$$

for each $g \in \mathcal{G}$. Thus, in this case, the proximal operator associated with the mixed ℓ_1/ℓ_2 -norm can be efficiently computed.

- Let $p = \infty$ so that Ω is mixed ℓ_1/ℓ_∞ -norm. In this case, for all $g \in \mathcal{G}$, the required orthogonal projection can be written as

$$\text{Proj}_{\mathcal{B}(\|\cdot\|_1, \lambda w_g)}(\mu_g) = \underset{v \in \mathbb{R}^{|g|}}{\text{argmin}} \frac{1}{2} \|v - \mu_g\|_2^2 \text{ subject to } \|v\|_1 \leq \lambda w_g.$$

Although this problem does not have a closed-form solution (unlike the previous example), there exist efficient methods for computing an accurate approximate solution [105, 106]. Once an approximate orthogonal projection is computed for each $g \in \mathcal{G}$, an approximate value for the associated proximal operator is obtained by combining (6.21) and Lemma 43.

Finally, we may now formulate the iterate update for the proximal gradient

method when solving problem (6.6). Specifically, we have

$$\begin{aligned}
x_{k+1} &= \operatorname{argmin}_{x \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^T(x - x_k) + \lambda\Omega(x) + \frac{1}{2\alpha_k}\|x - x_k\|_2^2 \\
&= \operatorname{argmin}_{x \in \mathbb{R}^n} \frac{1}{2\alpha_k}\|x - (x_k - \alpha_k \nabla f(x_k))\|_2^2 + \lambda\Omega(x) \\
&= \operatorname{Prox}_{\alpha_k \lambda \Omega}(x_k - \alpha_k \nabla f(x_k)).
\end{aligned} \tag{6.24}$$

For calculating (6.24), we may set $\mu \leftarrow x_k - \alpha_k \nabla f(x_k)$ and replace $\lambda \leftarrow \alpha_k \lambda$, and then apply (6.18), (6.19), and (6.21). For example, if $p = 2$ so that Ω is the mixed ℓ_1/ℓ_2 -norm, then it follows from (6.23) that

$$\begin{aligned}
[x_{k+1}]_g &= [\operatorname{Prox}_{\alpha_k \lambda \Omega}(x_k - \alpha_k \nabla f(x_k))]_g \\
&= \max \left\{ 0, 1 - \frac{\alpha_k \lambda w_g}{\|[x_k - \alpha_k \nabla f(x_k)]_g\|_2} \right\} \cdot [x_k - \alpha_k \nabla f(x_k)]_g
\end{aligned} \tag{6.25}$$

for all $g \in \mathcal{G}$. This update forms the basis for an extension of FaRSA for the mixed ℓ_1/ℓ_p -norm setting, which we discuss in the next section.

6.3 FaRSA for ℓ_1/ℓ_p -Regularization Problem

We now discuss the necessary adjustments to FaRSA to solve problem (6.6).

6.3.1 Optimality measure

The first change is the optimality measures for ℓ_1/ℓ_p -regularization problems. As shown in Lemma 12 for the ℓ_1 -regularization problem, the optimality measures $\|\beta(x_k)\|$ and $\|\phi(x_k)\|$ are essentially designed as the magnitude of the full ISTA step over the zero and non-zero variables, respectively. Recall that the full ISTA step is derived by the proximal operator for ℓ_1 -regularization in Section 3.1.1.1. Thus, it is natural to define the similar quantities for the ℓ_1/ℓ_p -

regularization problem that depend on the proximal operator associated with ℓ_1/ℓ_p -regularization. Specifically, given a partition \mathcal{G} of the variables, we define

$$[\beta(x_k)]_g = \begin{cases} [x_k - \text{Prox}_{\lambda\Omega}(x_k - \nabla f(x_k))]_g & \text{if } [x_k]_g = 0, \\ 0 & \text{otherwise,} \end{cases} \quad (6.26)$$

$$[\phi(x_k)]_g = \begin{cases} 0 & \text{if } [x_k]_g = 0, \\ [x_k - \text{Prox}_{\lambda\Omega}(x_k - \nabla f(x_k))]_g & \text{otherwise,} \end{cases} \quad (6.27)$$

for all $g \in \mathcal{G}$ so that

$$\beta(x_k) + \phi(x_k) = x_k - \text{Prox}_{\lambda\Omega}(x_k - \nabla f(x_k)). \quad (6.28)$$

In practice, we may use Lemma 43 to compute $\beta(x_k)$ and $\phi(x_k)$. For example, if Ω is the mixed ℓ_1/ℓ_2 -norm, then it follows from (6.25) that

$$[\text{Prox}_{\lambda\Omega}(x_k - \nabla f(x_k))]_g = \max \left(0, 1 - \frac{\lambda\omega_g}{\|[x_k - \nabla f(x_k)]_g\|_2} \right) [x_k - \nabla f(x_k)]_g$$

so that $\beta(x_k)$ and $\phi(x_k)$ can easily be computed in closed form.

6.3.2 Complete algorithm

We state the extension of FaRSA for the ℓ_1/ℓ_p -regularization problem as Algorithm 9. One can see that Algorithm 9 is similar to Algorithm 7, with two main differences. First, the definition of $\beta(x_k)$ and $\phi(x_k)$ is determined by the norm (regularizer) used, which for Algorithm 7 is the ℓ_1 -norm, and for Algorithm 9 is the mixed ℓ_1/ℓ_p -norm. Second, instead of choosing zero and nonzero variables for the index set \mathcal{I}_k as in Algorithm 7, in Algorithm 9 we choose subsets of the zero and nonzero *blocks* of variables to form \mathcal{I}_k . Finally, note that Algorithm 9

uses the same linesearch procedures as in Algorithm 7. Another possibility because of the use of the mixed ℓ_1/ℓ_p -norm, is to modify LINESEARCH_ϕ so that instead of using orthant-wise projections, it carefully handles the possibility that points of non-differentiability are encountered during the linesearch.

Algorithm 9 FaRSA for solving problem (6.6).

- 1: **Input:** x_0 and partition $\mathcal{G} = \{g_1, g_2, \dots, g_r\}$ of the variables $\{1, 2, \dots, n\}$.
- 2: **Constants:** $\{\eta_\phi, \eta_\beta\} \subset (0, 1]$, $\xi \in (0, 1)$, $\eta \in (0, 1/2]$, and $\{\gamma, \epsilon\} \subset (0, \infty)$
- 3: **for** $k = 0, 1, 2, \dots$ **do**
- 4: **if** $\max\{\|\beta(x_k)\|, \|\phi(x_k)\|\} \leq \epsilon$ **then**
- 5: **Return** the (approximate) solution x_k of problem (6.6).
- 6: **if** $\|\beta(x_k)\| \leq \gamma\|\phi(x_k)\|$ **then**
- 7: Choose $\mathcal{S} \subseteq \{1, 2, \dots, r\}$ such that, with $\mathcal{I}_k = \cup_{i \in \mathcal{S}} g_i$, it holds that

$$[x_k]_{g_i} \neq 0 \text{ for all } i \in \mathcal{S}, \text{ and } \|[\phi(x_k)]_{\mathcal{I}_k}\| \geq \eta_\phi \|\phi(x_k)\|.$$

- 8: Set $H_k \leftarrow \nabla_{\mathcal{I}_k \mathcal{I}_k}^2 F(x_k)$ and $g_k \leftarrow \nabla_{\mathcal{I}_k} F(x_k)$.
- 9: Compute the reference direction

$$d_k^R \leftarrow -\alpha_k g_k, \text{ where } \alpha_k \leftarrow \|g_k\|^2 / (g_k^T H_k g_k).$$

- 10: Compute any $\bar{d}_k \approx \text{argmin } m_k(d)$ (see (4.14)) such that

$$g_k^T \bar{d}_k \leq g_k^T d_k^R, \text{ and} \tag{6.29}$$

$$m_k(\bar{d}_k) \leq m_k(0) \tag{6.30}$$

- 11: Set $[d_k]_{\mathcal{I}_k} \leftarrow \bar{d}_k$ and $[d_k]_i \leftarrow 0$ for $i \notin \mathcal{I}_k$.
- 12: Use Algorithm 6 to compute $x_{k+1} \leftarrow \text{LINESEARCH}_\phi(x_k, d_k, \mathcal{I}_k, \eta, \xi)$.
- 13: **else**
- 14: Choose $\mathcal{S} \subseteq \{1, 2, \dots, r\}$ such that, with $\mathcal{I}_k = \cup_{i \in \mathcal{S}} g_i$, it holds that

$$[x_k]_{\mathcal{I}_k} = 0 \text{ and } \|[\beta(x_k)]_{\mathcal{I}_k}\| \geq \eta_\beta \|\beta(x_k)\|.$$

- 15: Set $[d_k]_{\mathcal{I}_k} \leftarrow -[\beta(x_k)]_{\mathcal{I}_k}$ and $[d_k]_i \leftarrow 0$ for $i \notin \mathcal{I}_k$.
 - 16: Use Algorithm 4 to compute $x_{k+1} \leftarrow \text{LINESEARCH}_\beta(x_k, d_k, \eta, \xi)$.
-

Chapter 7

FaRSA: Shared-Memory

Multi-core Parallelization

Given the explosion in size and complexity of modern datasets, it is increasingly important to build models with large numbers of features using large numbers of training examples. As a consequence, the need for scalable optimization is paramount since more applications use these high-dimensional datasets, e.g. building personalized feature-based recommendation system over historical information of billions of customers and product information [107]. Unfortunately, in recent years, the growth rate of a processor core speed is insufficient to handle the increasing requirements of modern applications. (According to Moore’s Law, the processing speed, or overall processing power for computers double every two years [108].) Instead, distributed systems of modern computers as well as computers with more cores have become popular (e.g., see Amazon AWS [109] and Microsoft Azure [110]). Hence, the need to utilize these parallel resources effectively within optimization algorithms has become a new challenge. Generally, comparing with serial optimization algorithms, parallel algorithms are more challenging in both theo-

retical and implementation aspects. Consequently, fewer parallel optimization algorithms exist compared to serial variants.

In this section, we show how to employ FaRSA in a parallel manner and compare it with state-of-the-art parallel optimization software for ℓ_1 -regularization problems. We organize the remainder of this chapter as follows. At first, we present the necessary background about parallel programming models in Section 7.1. In Section 7.2, we then briefly review related work about parallel algorithms for solving ℓ_1 -regularization problems. We next briefly discuss how to employ FaRSA in parallel in Section 7.3. Finally, we provide a numerical comparison between parallel FaRSA and state-of-the-art parallel optimization software in Section 7.4.

7.1 Background of Parallel Programming

In computing, a parallel programming model is an abstraction of a parallel computer architecture, which allows for a convenient way to express algorithms. The value of a programming model can be judged by its generality (how well a range of different problems can be expressed) and its performance (how efficiently the compiled programs can execute [111]). Generally, the classification of parallel programming models for optimization algorithms mainly focus on two concepts: process interaction and task decomposition.

7.1.1 Process interaction

Process interaction is the mechanism in which parallel processes communicate with each other. The most common forms of interaction are shared memory and message passing (distributed memory). We discuss these next.

- **Shared memory:** In shared-memory models, parallel processes share a

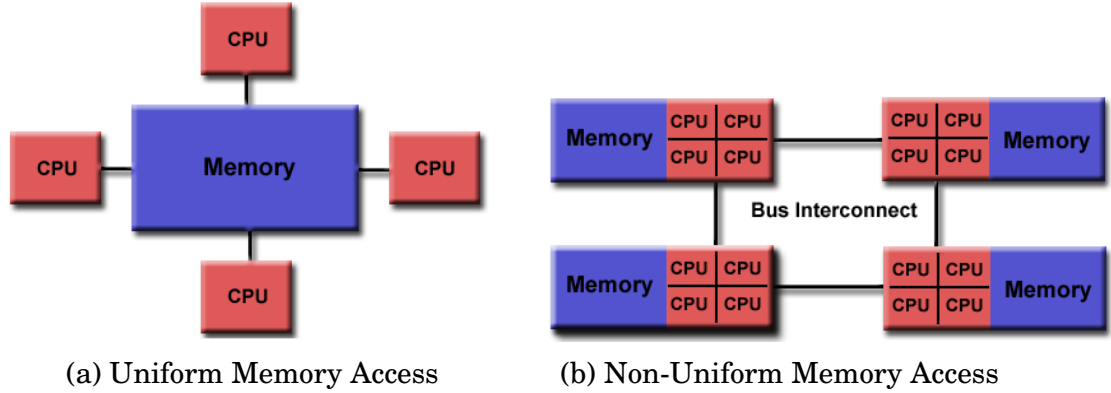


Figure 7.1: Underlying architectures for shared-memory model.

global address space that they read and write to asynchronously. Modern multi-core processors directly support shared memory, and many parallel programming languages and libraries, such as OpenMP (Open Multi-Processing) [112] and Cilk [113], are designed to exploit such a setup. Generally, there exist two memory access architectures, i.e. uniform memory access and non-uniform memory access, illustrated by Figure 7.1 from [114].

- **Message passing or distributed memory:** In a message-passing model, parallel processes usually do not possess an area of shared memory, but instead they exchange data by passing messages to each another. These communications can be asynchronous or synchronous. Building a reliable and flexible distributed system for efficient message passing is challenging. To avoid extra cost in building such a message passing system, many popular message passing interfaces have been designed for assisting users, e.g., Open MPI (Open Message Passing Interface) [115] and Intel MPI [116].

7.1.2 Task decomposition

Task decomposition relates to the way in which the constituent processes are organized. For mathematical optimization, there are two main categories: dis-

tributed optimization and optimization on shared-memory multi-core systems.

- **Distributed optimization:** In distributed optimization, processes are usually distinct. Each processor is able to access only a subset of data, is responsible for updating part of the variables based on local information, and emphasizes the need for communication. Different processes usually require message passing to communicate, which makes communicating efficiently an important property for distributed optimization algorithm [117, Chapter 10].

- **Optimization on shared-memory multi-core systems:** This type of optimization algorithm focuses on organizing processes to complete each single operation by cooperation. The data address is shared among all processors, and tasks are performed independently on disjoint groups of data, where each group of data is assigned to one of the available processors.

7.2 Prior Parallel Algorithms for ℓ_1 -Problems

Scherrer presented an abstract framework for coordinate descent methods suitable for a parallel computing environment [118, 119]. Several coordinate descent algorithms, such as SHOTGUN [120] and GROCK [121], can be covered by this framework. Briefly, the framework converts problem (1.1) to a smooth version, and then during each iteration SHOTGUN and GROCK randomly select a set of coordinates, distribute them to multiple-processors, and then update the variables independently via a projected gradient calculation. The difference between the two is that SHOTGUN uniformly selects the coordinates to be updated, while GROCK uses gradient information to do a greedy selection. SHOTGUN and GROCK are relatively easy to implement, but may not converge in practice due to their non-monotone nature (no line search is used).

Jaggi proposed a distributed method called COCOA [122], which is a primal-dual framework. The key idea of COCOA is to convert the primal problem into its corresponding dual problem, and then solve its dual. However, this primal-dual method is only equipped to handle strongly convex regularizers, thus making it not suitable for the ℓ_1 -regularizer. Smith proposes a framework (PROXCOCOA [123]) that can run either on the dual or the primal directly.

Some of the state-of-the-art serial algorithms discussed in Chapter 3 also have available parallel versions. Parallel OWL-QN (Orthant-Wise Limited-memory Quasi-Newton) is implemented in Apache Spark’s MLlib (v1.5.0) [124]. Multi-core LIBLINEAR is built on a shared-memory multi-core system using OpenMP [125]. Given sufficient memory, it appears that multi-core LIBLINEAR is the most commonly used among the reviewed parallel solvers.

In general, even though there exist many sequential optimization algorithms for the ℓ_1 -regularization problem described in Chapter 3, many fewer parallel algorithms exist. Comparing with sequential optimization algorithms, parallel algorithms are more challenging in both theoretical and implementation aspects based on the different ways of parallelization. For distributed algorithms, more limited convergence results can be obtained since local progress in optimization does not typically guarantee global progress. For algorithms built upon a shared-memory multi-core system, usually no difference in theoretical aspects exist between serial and parallel algorithms because of the nature of multi-core shared-memory parallelization (see Section 7.1.2).

7.3 FaRSA: Multi-Core and Shared-Memory

Following the development of random-access memory (RAM), shared-memory systems such as the common desktop now possess more memory (e.g. 16GB

or 32GB of main memory) to store data. Also, modern computers now contain multi-core CPUs or GPUs for addressing vast threads of tasks. The increasing capability of memory and CPUs allows optimization algorithm to be further accelerated. Yet, it is still the case that solving optimization problems can be slow if the power of multi-core CPUs is not fully utilized. Therefore, designing effective parallel methods is crucial to achieve significant acceleration [126].

For meeting the increasing requirements of scalable optimization, we have developed a parallel version of FaRSA for shared-memory and multi-core systems. Rather than designing distributed version of algorithm by editing our serial algorithm shown in Algorithm 7, here we parallelize the computations of FaRSA using multiple threads, e.g. to compute matrix-vector multiplications. The easiest way to accelerate FaRSA is to parallelize the loops using OpenMP [112], which is simple and common in shared-memory systems. Among the toolkits of OpenMP, its `parallel for` loop is the main component that we utilize.

The numerical results that we present indicate that, under suitable conditions, a significant speedup can be achieved by FaRSA on a shared-memory and multi-core system. It is also interesting that parallelization by OpenMP does not always mean acceleration, because performance may suffer from the following three factors:

- **Atomic operations:** Multi-core FaRSA includes some atomic operations that avoid the need for threads to update the same variables simultaneously. These atomic operations are relatively expensive.
- **Synchronization:** A lot of synchronization is used to ensure the reliability of the multi-core parallelization. The synchronization mechanisms of OpenMP sacrifice the speed of the whole process.

- **Computation Precision:** In OpenMP, a single `for loop` is first decomposed into a few `sub for loops` for each processor, and the results are then aggregated together. The precision of processing the `for loop` serially may be different from that of parallelizing the `for loop`, since the order of computations matters in some scenarios. For example, suppose we utilize OpenMP with a `reduction` clause to compute the sum of the entries in an array representing $x = (x_1, x_2, \dots, x_m)$. In the serial case, one processor computes $x_1 + x_2 + \dots + x_m$ by taking additions in the ascending order of indices. In the parallel case with static scheduling, the i th processor computes $x_{i_1} + x_{i_2} + \dots + x_{i_{m/P}}$. Since floating-point addition is a non-associative operation due to round-off errors, these two ways may produce different results.

7.4 Numerical Experiments

In this section, we test the performance of FaRSA on a multi-core shared-memory system. We describe the experimental setup, show numerical results, and compare our results to state-of-the-art methods on multi-core shared-memory system for ℓ_1 -regularization problems (i.e. multi-core LIBLINEAR).

7.4.1 Experimental setup

The test problem is still the ℓ_1 -regularized logistic regression problem (1.1) for binary classification with f defined in (1.6) and $\lambda = 1/N$ the weighting parameter in (1.1). We tested multi-core FaRSA on different tolerance levels, namely 10^{-2} , 10^{-4} and 10^{-6} . Since the motivation of parallelization is to solve problems based on high-dimensional datasets, we selected the largest seven data sets in Table 5.2, namely *url_combined*, *SUSY*, *avazu-app.tr*, *kdda*, *kddb*, *HIGGS*, and *epsilon* for our test set. All experiments are conducted

Table 7.1: The final objective function value for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-6}$.

dataset	1-core	2-cores	3-cores	4-cores	5-cores	6-cores	7-cores	8-cores
url.combined	0.02624	0.02624	0.02624	0.02624	0.02624	0.02624	0.02624	0.02624
SUSY	0.46300	0.46300	0.46300	0.46300	0.46300	0.46300	0.46300	0.46300
avazu-app.tr	0.30009	0.30009	0.30009	0.30009	0.30009	0.30009	0.30009	0.30009
kdda	—	—	—	—	—	—	—	—
kddb	—	—	—	—	—	—	—	—
HIGGS	0.63771	0.63771	0.63771	0.63771	0.63771	0.63771	0.63771	0.63771
epsilon	0.25845	0.25845	0.25845	0.25845	0.25845	0.25845	0.25845	0.25845

Table 7.2: The number of iterations for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-6}$.

dataset	1-core	2-cores	3-cores	4-cores	5-cores	6-cores	7-cores	8-cores
url.combined	139	167	145	158	220	125	188	173
SUSY	34	34	34	34	34	34	34	34
avazu-app.tr	68	86	69	70	72	77	64	72
kdda	—	—	—	—	—	—	—	—
kddb	—	—	—	—	—	—	—	—
HIGGS	13	13	13	13	13	13	13	13
epsilon	33	33	34	34	33	34	34	34

using the cluster at the Computational Optimization Research laboratory at Lehigh University (COR@L) with 30GB of main memory and eight 2.0GHz AMD CPUs. For comparison purposes, we downloaded the latest version of multi-core LIBLINEAR (version 2.20) from <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/multicore-liblinear>.

7.4.2 Numerical results for multi-core FaRSA

Tight tolerance: $\text{tol} = 10^{-6}$. First, we investigate the performance of FaRSA on a multi-core shared-memory system with various tolerance levels. For the tight and default tolerance level 10^{-6} , Table 7.1 shows that FaRSA can reach the same final objective function values (to five digits of accuracy) for one to eight cores. However, this does not mean that FaRSA performs exactly the same in terms of iterations since it is shown in Table 7.2 that, for datasets *url.combined* and *avazu-app.tr*, FaRSA with a different number of cores solves them with a different number of iterations. This may be caused by the precision issues associated with multi-core parallelization as described in Section 7.3.

In terms of speedup, the atomic operations and synchronization requirements mean that the speedup in the multi-core setting is not quite linear (see Figure 7.2), where the speedup is defined as

$$\text{speedup} = \frac{\text{wall time of FaRSA with multiple cores}}{\text{wall time of FaRSA with single core}}. \quad (7.1)$$

In general, the speedup is smaller than the number of cores. For *epsilon*, the speedup increases roughly linearly as the number of cores increases. For *SUSY* and *HIGGS*, the speedup increases dramatically when the number of cores are relatively small, and then slows down as the number of cores continues to increase. For *url_combined* and *avazu-app.tr*, even though significant acceleration can be achieved through multi-core parallelization, we observe that high fluctuation occurs, which is consistent with the fluctuation on the number of iterations shown in Table 7.2. This effect appears to be caused mainly by the precision issue of multi-core parallelization.

We now consider a looser termination tolerance.

Weaker termination tolerance: $\text{tol} = 10^{-4}$. We now illustrate the performance of multi-core FaRSA when a weaker termination tolerance is used. The results may be found in Table 7.3, Table 7.4, and Figure 7.3.

We first observe from Table 7.3 that under the looser termination tolerance of 10^{-4} , the two problems *kdda* and *kddb* are able to be solved, which was not true for FaRSA when the termination tolerance 10^{-6} was used. Moreover, it is shown in Figure 7.3 that for datasets *SUSY*, *HIGGS*, and *epsilon*, the speedups are less than the number of cores, and increases roughly linearly as the number of cores increases. On the other hand, for datasets *url_combined*, *avazu-app.tr*, and *kdda*, acceleration is achieved by using multi-core parallelization but with significant fluctuations. Note that no speedup for *kddb* is provided because

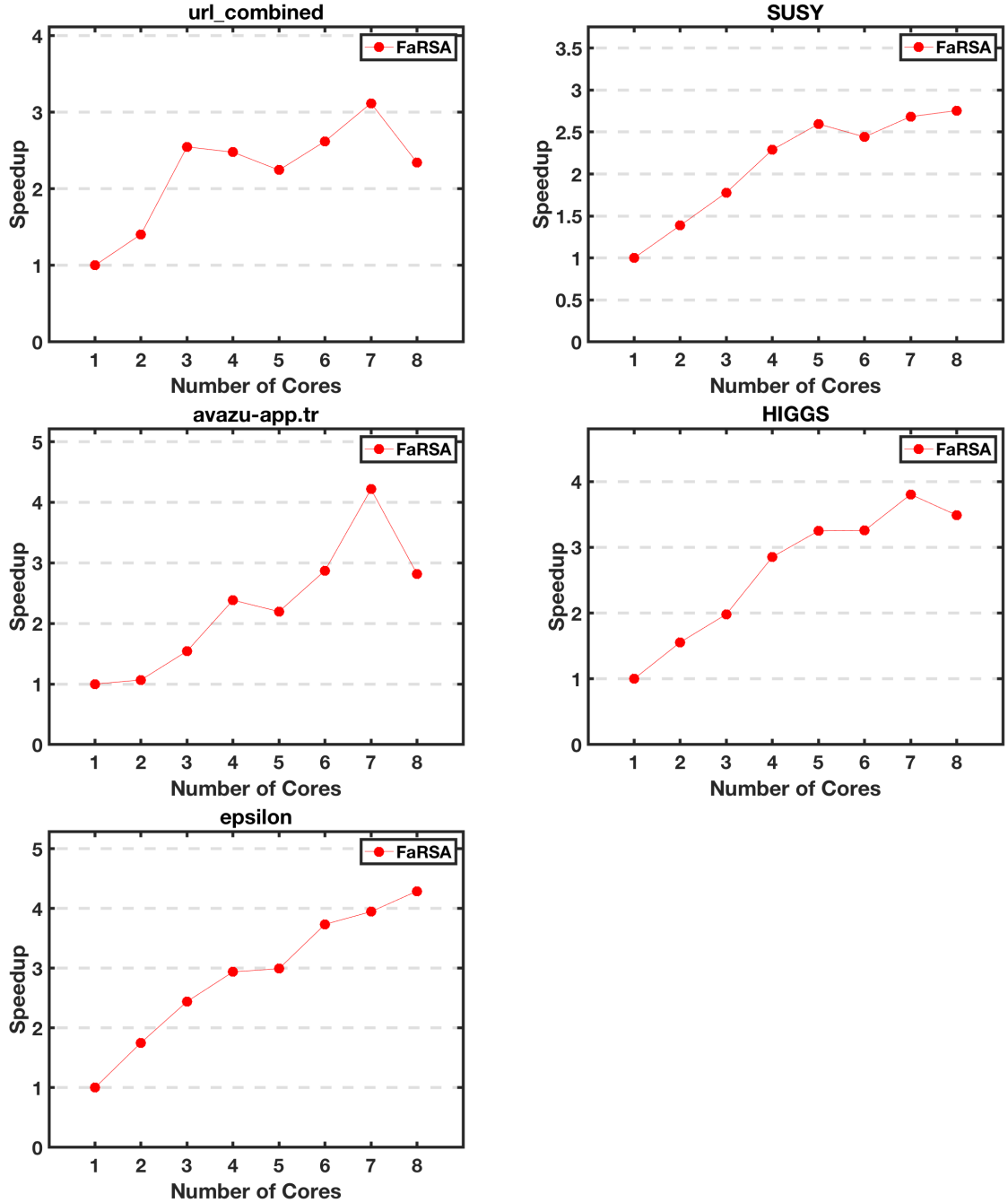


Figure 7.2: Speedup of FaRSA for termination tolerance level $\text{tol} = 10^{-6}$.

FaRSA fails when a single core is used. In Table 7.4 we may still observe the appearance of different number of iterations under multi-core settings because of the precision feature (e.g., for *url combined*, *avazu-app.tr*, *kdda*, and *kddb*).

We now test an even weaker termination tolerance.

Table 7.3: The final objective function value for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-4}$

dataset	1-core	2-cores	3-cores	4-cores	5-cores	6-cores	7-cores	8-cores
url_combined	0.02728	0.03048	0.02982	0.02776	0.02762	0.02796	0.02791	0.02796
SUSY	0.46302	0.46302	0.46302	0.46302	0.46302	0.46302	0.46302	0.46302
avazu-app.tr	0.30063	0.30034	0.30058	0.30040	0.30034	0.30026	0.30026	0.30034
kdda	0.26452	0.26476	0.26448	0.26460	0.26462	0.26452	0.26474	0.26445
kddb	—	—	0.25172	0.25179	0.25200	0.25180	0.25162	0.25161
HIGGS	0.63771	0.63771	0.63771	0.63771	0.63771	0.63771	0.63771	0.63771
epsilon	0.25864	0.25864	0.25864	0.25864	0.25864	0.25864	0.25864	0.25864

Table 7.4: The number of iterations for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-4}$.

dataset	1-core	2-cores	3-cores	4-cores	5-cores	6-cores	7-cores	8-cores
url_combined	88	67	70	59	94	68	80	78
SUSY	26	26	26	26	26	26	26	26
avazu-app.tr	15	15	15	17	15	18	18	15
kdda	40	37	39	32	32	36	37	35
kddb	—	—	32	27	30	43	34	36
HIGGS	12	12	12	12	12	12	12	12
epsilon	25	25	25	25	25	25	25	25

Table 7.5: The final objective function value for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-2}$.

dataset	1-core	2-cores	3-cores	4-cores	5-cores	6-cores	7-cores	8-cores
url_combined	0.08380	0.08380	0.08380	0.08380	0.08380	0.08380	0.08380	0.08380
SUSY	0.47215	0.47215	0.47215	0.47215	0.47215	0.47215	0.47215	0.47215
avazu-app.tr	0.35026	0.35026	0.35026	0.35026	0.35026	0.35026	0.35026	0.35026
kdda	0.30837	0.30837	0.30837	0.30837	0.30837	0.30837	0.30837	0.30837
kddb	0.32486	0.32486	0.32486	0.32486	0.32486	0.32486	0.32486	0.32486
HIGGS	0.63839	0.63839	0.63839	0.63839	0.63839	0.63839	0.63839	0.63839
epsilon	0.30670	0.30670	0.30670	0.30670	0.30670	0.30670	0.30670	0.30670

Table 7.6: The number of iterations for FaRSA using different numbers of cores on the big datasets in Table 5.2 with termination tolerance $\text{tol} = 10^{-2}$.

dataset	1-core	2-cores	3-cores	4-cores	5-cores	6-cores	7-cores	8-cores
url_combined	10	10	10	10	10	10	10	10
SUSY	6	6	6	6	6	6	6	6
avazu-app.tr	3	3	3	3	3	3	3	3
kdda	8	8	8	8	8	8	8	8
kddb	6	6	6	6	6	6	6	6
HIGGS	8	8	8	8	8	8	8	8
epsilon	8	8	8	8	8	8	8	8

Weakest termination tolerance: $\text{tol} = 10^{-2}$. We now test FaRSA under the looser termination tolerance, namely $\text{tol} = 10^{-2}$. The results of these tests may be seen in Table 7.5, Table 7.6, and Figure 7.4.

As may be seen in Table 7.6, the number of iterations required to solve all test problems is now the same across all number of cores used, and from

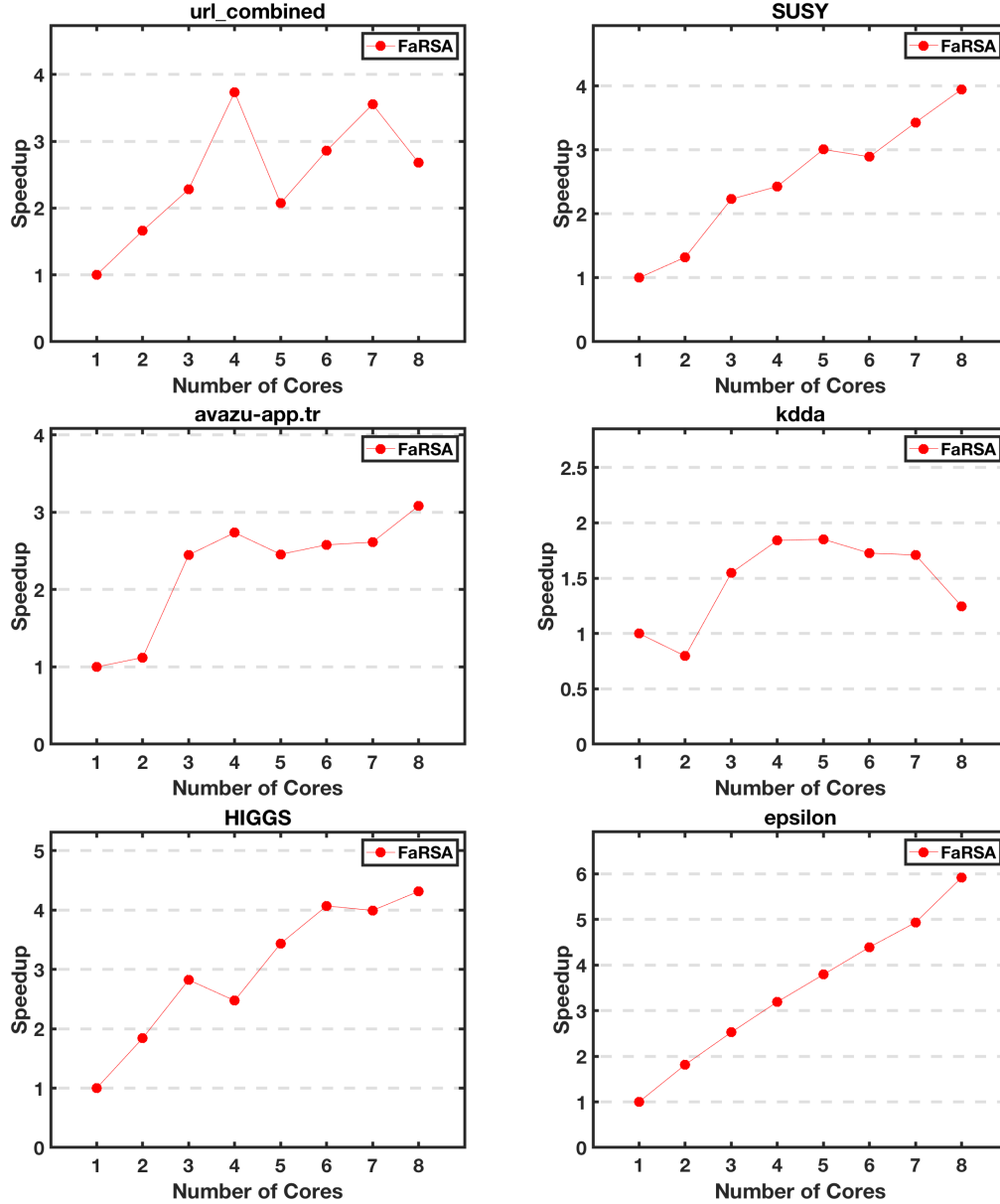


Figure 7.3: Speedup of FaRSA for termination tolerance level $\text{tol} = 10^{-4}$.

Table 7.5 the same is true for the final objective value obtained. These results for the number of iterations and final objective function values indicate that the precision issue associated with multi-core parallelization does not have a significant effect when low-accuracy is requested. Also, from Figure 7.4, it is interesting that the speedup is consistently roughly linear in the number of cores (with some occasional fluctuation). The three test problems that did not

achieve linear acceleration under tighter tolerances, i.e. *url_combined*, *kdda*, and *kddb*, achieve smaller speedups than the other problems.

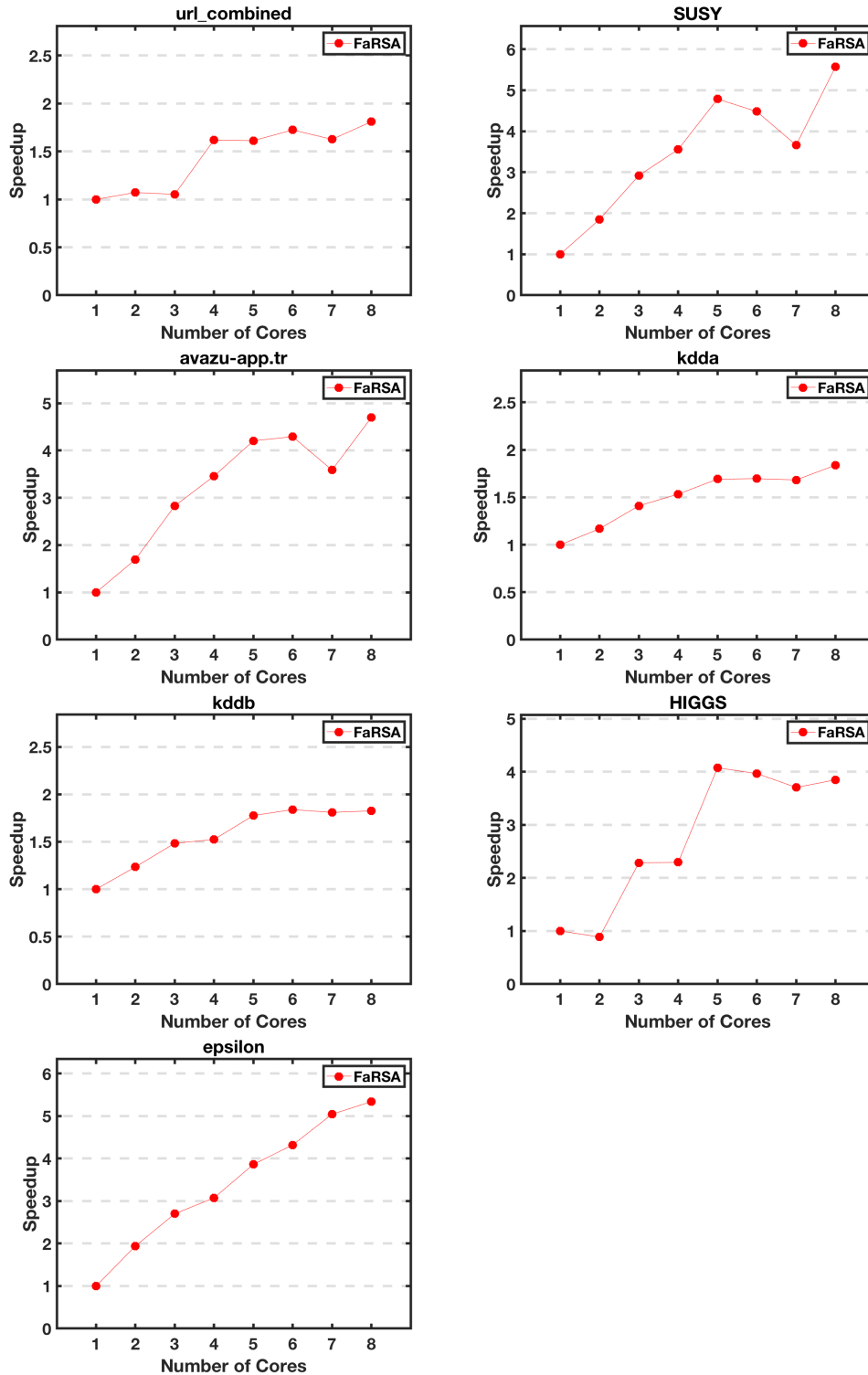


Figure 7.4: Speedup of FaRSA for termination tolerance level $\text{tol} = 10^{-2}$.

7.4.3 Multi-core FaRSA versus multi-core LIBLINEAR

We now compare multi-core FaRSA and multi-core LIBLINEAR under different tolerances. The experimental setup is the same as Section 7.4.1. As with the experiments shown in Chapter 5, we focus on comparing the two main evaluation metrics for optimization algorithms, namely the wall times for solving the problems and the final computed objective values.

Tight tolerance: $\text{tol} = 10^{-6}$. Figure 7.5 and Figure 7.6 contain the results for multi-core FaRSA and multi-core LIBLINEAR when a stopping tolerance of 10^{-6} is used. We may see that both algorithms benefit from multi-core acceleration, FaRSA spends less time solving these big data problems, and both algorithms achieve approximately the same final objective function values on the problems that are successfully solved. For dataset *avazu-app.tr*, LIBLINEAR does not terminate within the maximum allowed wall time even with multi-core acceleration. Both methods cannot solve *kdda* and *kddb* with $\text{tol} = 10^{-6}$.

Weaker tolerance: $\text{tol} = 10^{-4}$. The results for this termination tolerance may be found in Figure 7.7 and Figure 7.8. It is still the case that FaRSA spends less time than LIBLINEAR to solve the problems. LIBLINEAR is now able to solve *kddb* when at least seven cores are used, whereas FaRSA is able to solve with only three cores. Both methods achieve similar final object values, except for dataset *url_combined* where fluctuations occur for FaRSA.

Weakest tolerance: $\text{tol} = 10^{-2}$. The results for this termination tolerance may be found in Figure 7.9 and Figure 7.10. It is still the case that FaRSA needs less time to solve most of these big problems (exceptions are *kdda* and *url_combined*). However, LIBLINEAR performs better than FaRSA on *url_combined* in both run time and final objective function value aspects. For *kdda* both run time curves are tangled together, although LIBLINEAR can reach smaller objective values. For *SUSY*, *avazu-app.tr*, *kddb*, and *epsilon*, LI-

BLINEAR is much slower but achieve smaller objective values. Both FaRSA and LIBLINEAR achieve similar objective function value on dataset *HIGGS*.

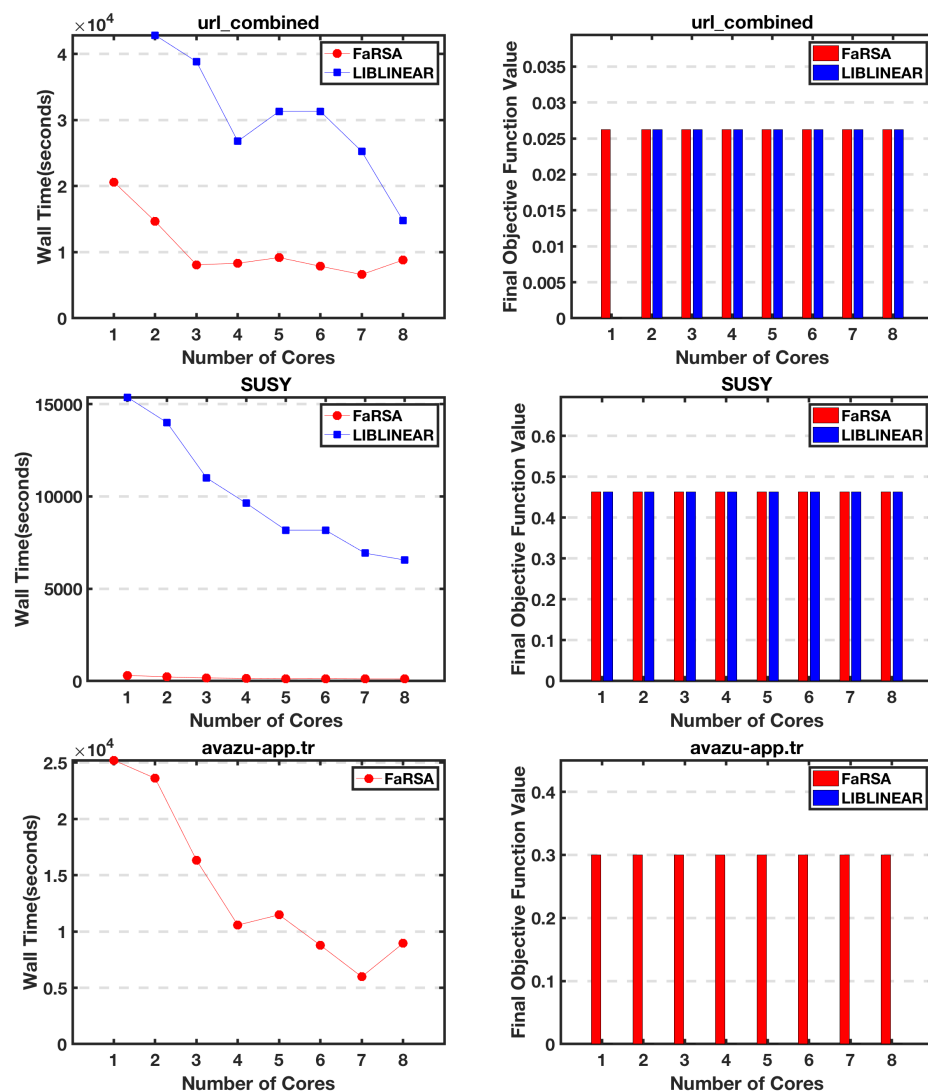


Figure 7.5: Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time (seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-6}$ and using different numbers of cores.

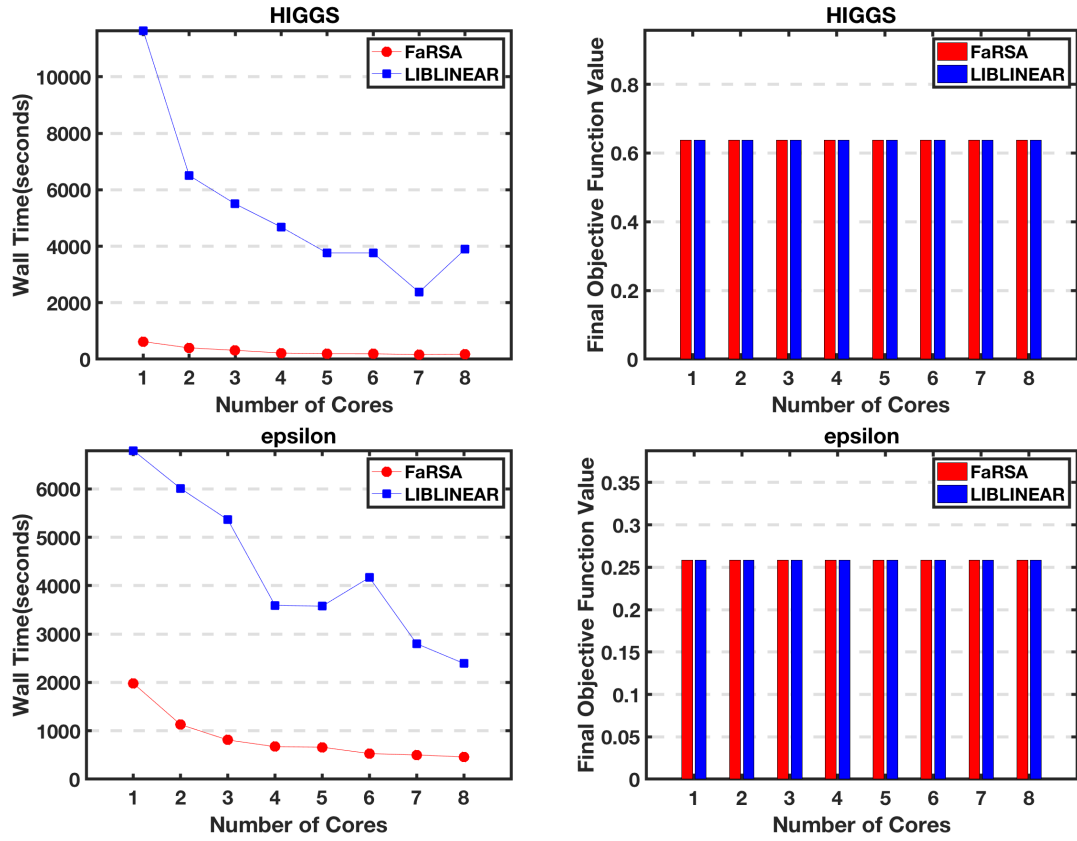


Figure 7.6: Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time (seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-6}$ and using different numbers of cores.

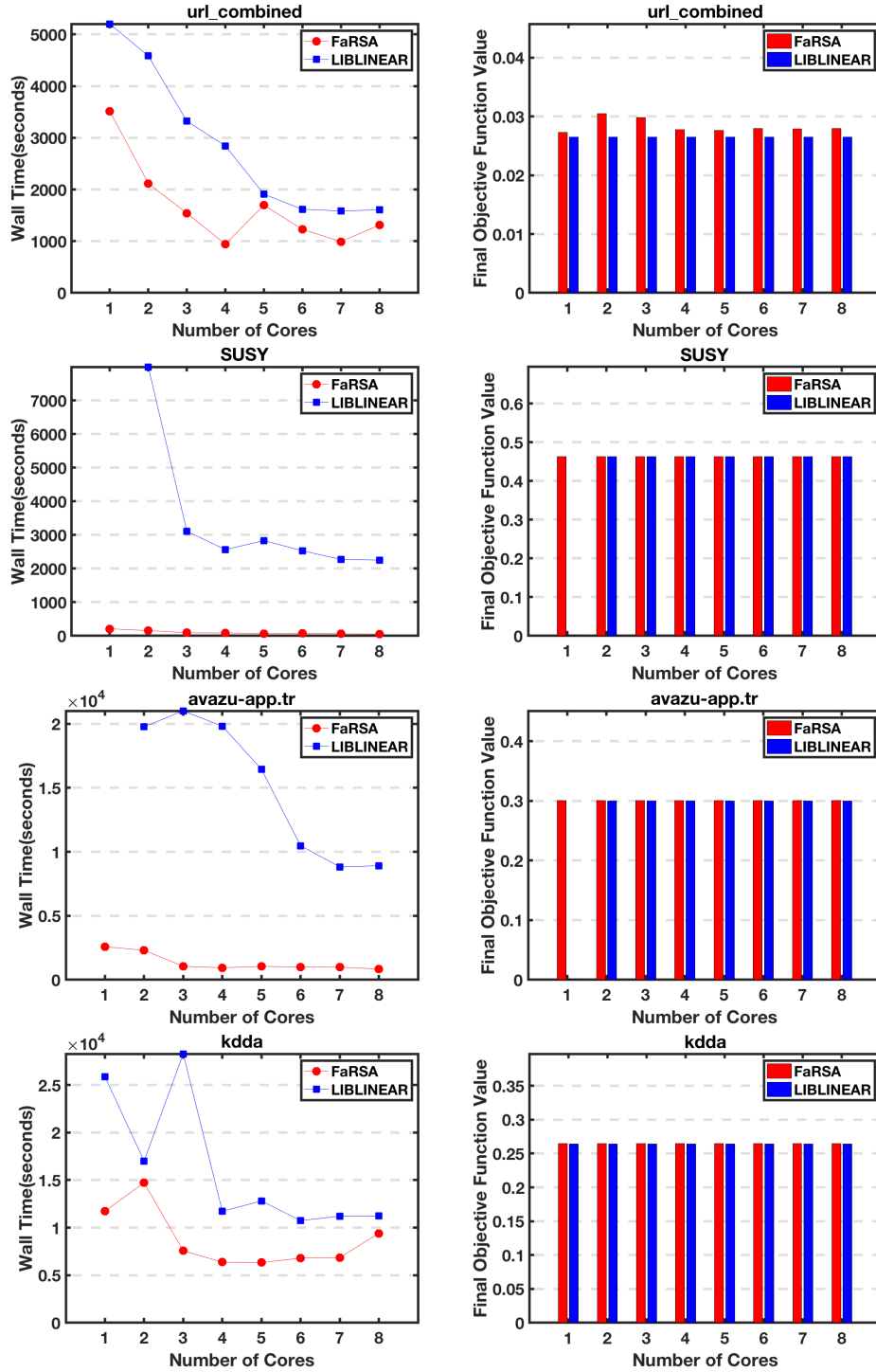


Figure 7.7: Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time(seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-4}$ and using different numbers of cores.

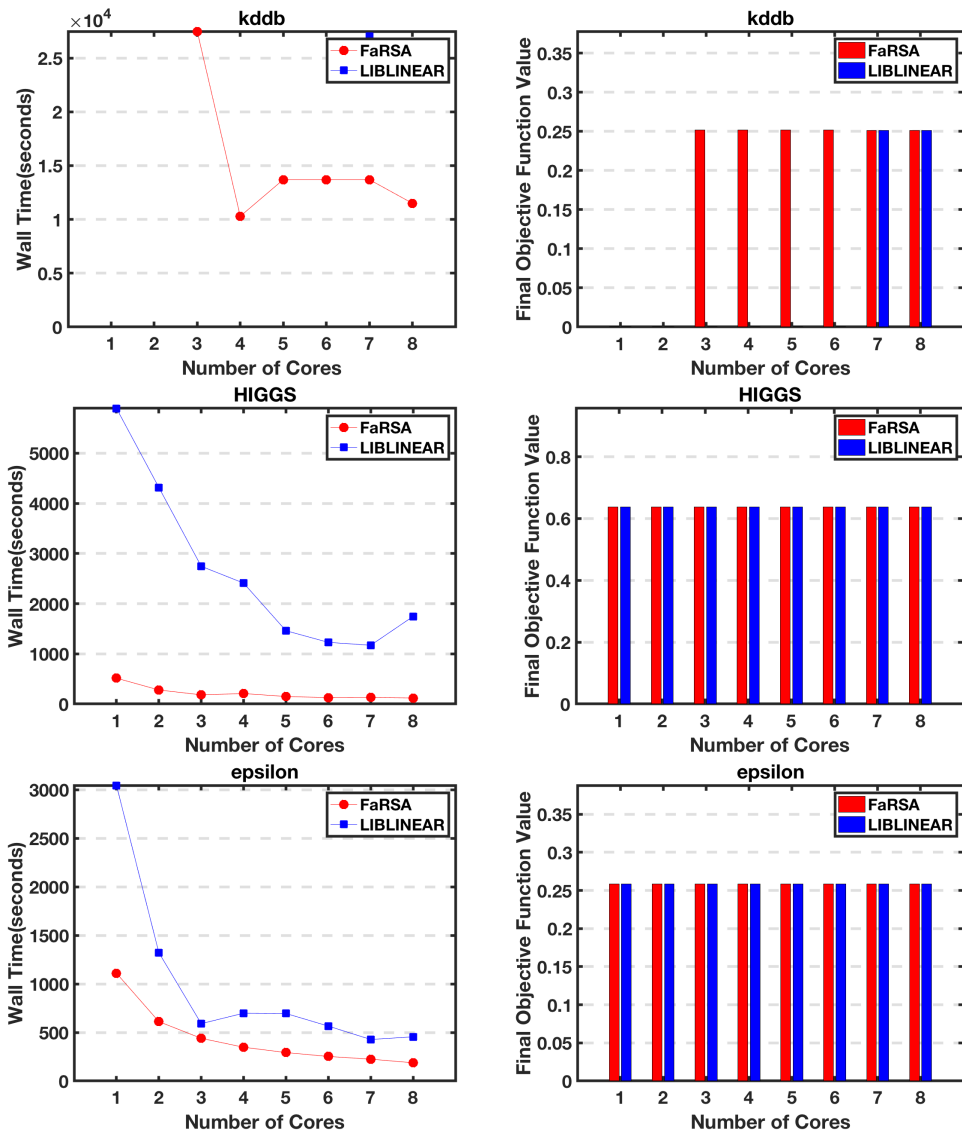


Figure 7.8: Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time(seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-4}$ and using different numbers of cores.

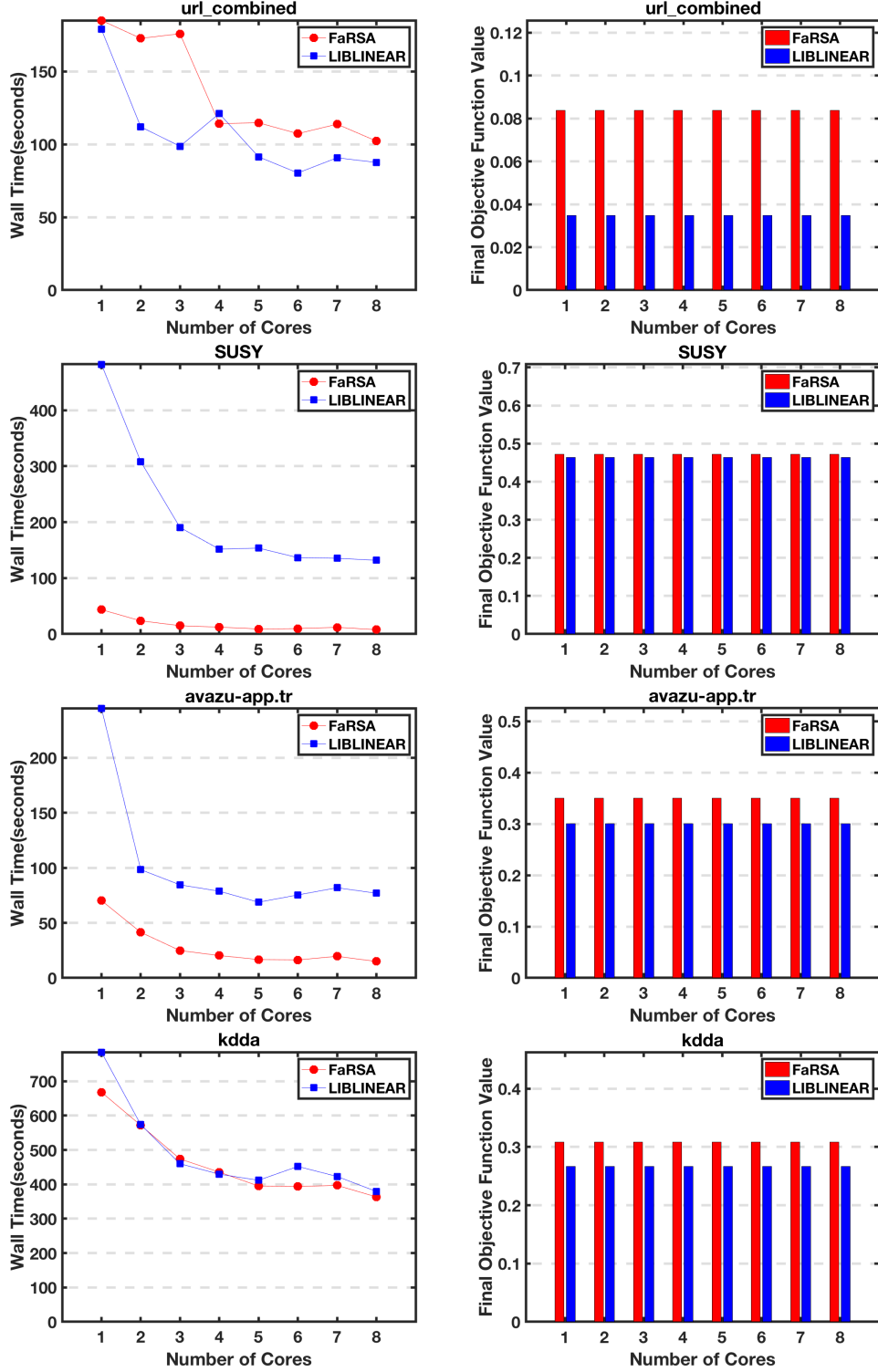


Figure 7.9: Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of on wall time (seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-2}$ and using different numbers of cores.

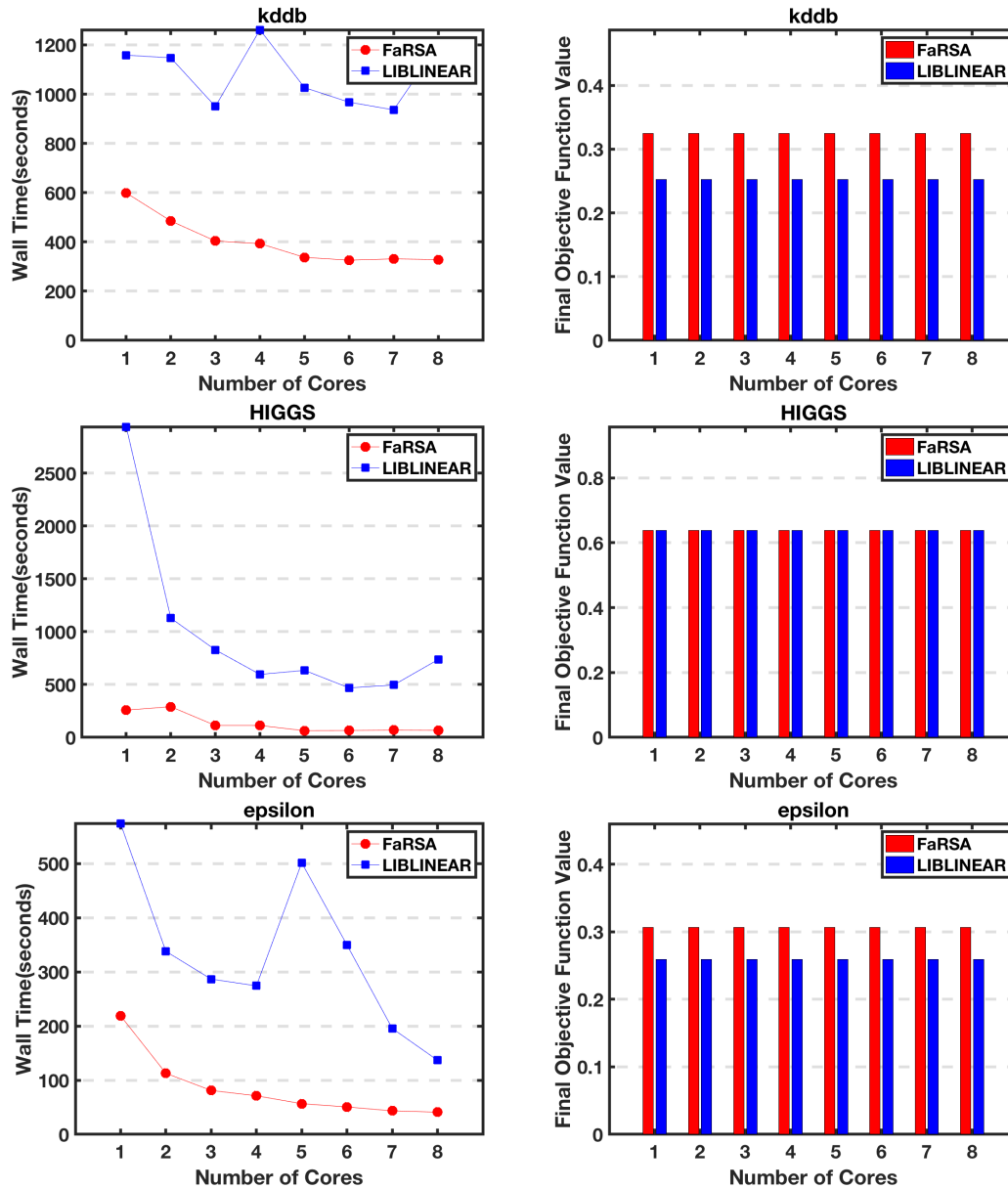


Figure 7.10: Comparison of multi-core FaRSA versus multi-core LIBLINEAR in terms of wall time (seconds) and final objective function values, with a termination tolerance of $\text{tol} = 10^{-2}$ and using different numbers of cores.

Chapter 8

Conclusion

In Chapter 4 we presented a framework, called FaRSA, for minimizing the sum of a convex function and an ℓ_1 -regularization function. We established global and local superlinear convergence results. Numerical results illustrated that our framework is efficient and robust when solving binary classification problems using an ℓ_1 -regularized logistic function approach over a diverse collection of datasets. The efficiency of our method is the result of three main features. The first feature is a simple strategy for predicting those variables that are zero at a solution. The second feature is a subspace minimization strategy that makes use of second-order derivatives, thus allowing for rapid convergence. The third feature is an effective strategy for determining during each iteration whether the variables predicted to be zero should be updated or whether additional subspace minimization should be performed.

In Chapter 5 we presented numerical results showing that our Matlab and C implementations of FaRSA outperform state-of-the-art algorithms. In general, our method outperforms first-order methods in terms of the number of iterations required to achieve the same accuracy. Relative to the first- and second-order methods, our method typically requires less computational time.

We believe that FaRSA must now be considered a state-of-the-art algorithm for ℓ_1 -regularization problems.

Although the thesis focused on ℓ_1 -regularization problems, in Chapter 6 we discussed how structured-sparsity problems can be handled by extending our proposed framework. Although such extensions are not currently implemented in our software, they will be included in a future release.

In Chapter 7, we further develop a parallel FaRSA on multi-core shared memory system to meet the rapidly increasing requirements of scalable optimization. The results of numerical experiments indicate that under proper conditions, FaRSA can achieve prominent acceleration by multi-core parallelization. Moreover, the comparison with state-of-the-art multi-core software LIBLINEAR shows that multi-core FaRSA outperforms multi-core LIBLINEAR generally, which implies that multi-core FaRSA is a state-of-the-art software on multi-core shared-memory system for ℓ_1 -regularization problem.

Bibliography

- [1] Stephen Bradley, Arnoldo Hax, and Thomas Magnanti. Applied mathematical programming. 1977.
- [2] Avinash K. Dixit. *Optimization in economic theory*. Oxford University Press on Demand, 1990.
- [3] Salah Haggag, Fatma Desokey, and Moutaz Ramadan. A cosmological inflationary model using optimal control. *Gravitation and Cosmology*, 23(3):236–239, 2017.
- [4] A. Parkinson, R. Balling, and J. Hedengren. *Optimization methods for engineering design*. Provo, UT: Brigham Young University., 2013.
- [5] Stephen Wright and Jorge Nocedal. *Numerical optimization*. Springer, 2006.
- [6] A. N. Tikhonov. Resolution of ill-posed problems and the regularization method (in russian). *Dokl. Akad.*, page 501–504, 1963.
- [7] A. N. Tikhonov and V. Y. Arsenin. Solution of ill-posed problems. *Washington: Winston and Sons.*, 1977.
- [8] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

- [9] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Structured sparsity through convex optimization. *Statistical Science*, pages 450–468, 2012.
- [10] Rodolphe Jenatton, Jean-Yves Audibert, and Francis Bach. Structured variable selection with sparsity-inducing norms. *Journal of Machine Learning Research*, 12(Oct):2777–2824, 2011.
- [11] M. Bishop Christopher. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [12] Curtis R. Vogel. *Computational methods for inverse problems*, volume 23. Siam, 2002.
- [13] Wenjiang J. Fu. Penalized regressions: the bridge versus the lasso. *Journal of computational and graphical statistics*, 7(3):397–416, 1998.
- [14] Stefan Riezler and Alexander Vasserman. Incremental feature selection and ℓ_1 -regularization for relaxed maximum-entropy modeling. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.
- [15] Francis Bach, Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, et al. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.
- [16] Maria D. Gonzalez-Lima, William W. Hager, and Hongchao Zhang. An affine-scaling interior-point method for continuous knapsack constraints with application to support vector machines. *SIAM Journal on Optimization*, 21(1):361–390, 2011.

- [17] N. Keskar, Jorge Nocedal, Figen Öztoprak, and Andreas Waechter. A second-order method for convex l1-regularized optimization with active-set prediction. *Optimization Methods and Software*, 31(3):605–621, 2016.
- [18] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. An improved glmnet for l1-regularized logistic regression. *Journal of Machine Learning Research*, 13(Jun):1999–2030, 2012.
- [19] Galen Andrew and Jianfeng Gao. Scalable training of l1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. ACM, 2007.
- [20] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.
- [21] T. Strutz. *Data Fitting and Uncertainty*. Springer, 2016.
- [22] Abraham Charnes, Edward L. Frome, and Po-Lung Yu. The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. *Journal of the American Statistical Association*, 71(353):169–171, 1976.
- [23] Peter Bühlmann and Sara Van De Geer. *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media, 2011.
- [24] Tikhonov A.N., Goncharsky A.V., Stepanov V.V., and Yagola A.G. Numerical methods for the solution of ill-posed problems. *Kluwer Academic Publishers*, 1995.

- [25] Tikhonov A.N., Leonov A.S., and Yagola A.G. Nonlinear ill-posed problems. *Chapman and Hall.*, 1998.
- [26] Statistics 305 Stanford. Regularization: Ridge regression and the lasso.
- [27] Leo Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):373–384, 1995.
- [28] Xiaoming Huo and Xuelei Ni. When do stepwise algorithms meet subset selection criteria? *The Annals of Statistics*, pages 870–887, 2007.
- [29] Ehsan Elhamifar and René Vidal. Sparse subspace clustering. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2790–2797. IEEE, 2009.
- [30] Congyuan Yang, Daniel P. Robinson, and Rene Vidal. Sparse subspace clustering with missing entries. In *International Conference on Machine Learning*, pages 2463–2472, 2015.
- [31] Chong You, Chun-Guang Li, Daniel P. Robinson, and René Vidal. Oracle based active set algorithm for scalable elastic net subspace clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3928–3937, 2016.
- [32] Chong You, Daniel P. Robinson, and René Vidal. Scalable sparse subspace clustering by orthogonal matching pursuit. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3918–3927, 2016.
- [33] Adam L. Berger. A brief maxent tutorial. 1996.

- [34] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.
- [35] Stephen J. Wright, Robert D. Nowak, and Mário AT Figueiredo. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493, 2009.
- [36] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyu Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.
- [37] Frank E. Curtis, Zheng Han, and Daniel P. Robinson. A globally convergent primal-dual active-set framework for large-scale convex quadratic optimization. *Computational Optimization and Applications*, 60(2):311–341, 2015.
- [38] William W. Hager and Hongchao Zhang. A new active set algorithm for box constrained optimization. *SIAM Journal on Optimization*, 17(2):526–557, 2006.
- [39] M. Ko, Jochem Zowe, et al. An iterative two-step algorithm for linear complementarity problems. *Numerische Mathematik*, 68(1):95–106, 1994.
- [40] Chih-Jen Lin and Jorge J. Moré. Newton’s method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.
- [41] Hassan Mohy-ud Din and Daniel P. Robinson. A solver for nonconvex bound-constrained quadratic optimization. *SIAM Journal on Optimization*, 25(4):2385–2407, 2015.

- [42] Jorge J. Moré and Gerardo Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, 1(1):93–113, 1991.
- [43] Daniel P. Robinson, Liming Feng, Jorge M. Nocedal, and Jong-Shi Pang. Subspace accelerated matrix splitting algorithms for asymmetric and symmetric linear complementarity problems. *SIAM Journal on Optimization*, 23(3):1371–1397, 2013.
- [44] Richard H. Byrd, Jorge Nocedal, and Figen Oztoprak. An inexact successive quadratic approximation method for ℓ_1 -regularized optimization. *Mathematical Programming*, 157(2):375–396, 2016.
- [45] Cho-Jui Hsieh, Inderjit S. Dhillon, Pradeep K. Ravikumar, and Mátyás A Sustik. Sparse inverse covariance matrix estimation using quadratic approximation. In *Advances in neural information processing systems*, pages 2330–2338, 2011.
- [46] Jason D. Lee, Yuekai Sun, and Michael Saunders. Proximal newton-type methods for convex optimization. In *Advances in Neural Information Processing Systems*, pages 827–835, 2012.
- [47] Katya Scheinberg and Xiaocheng Tang. Practical inexact proximal quasi-newton method with global complexity analysis. *Mathematical Programming*, 160(1-2):495–529, 2016.
- [48] Richard H. Byrd, Gillian M. Chin, Jorge Nocedal, and Figen Oztoprak. A family of second-order methods for convex ℓ_1 -regularized optimization. *Mathematical Programming*, 159(1-2):435–467, 2016.
- [49] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Pret-

- tenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
- [50] Amazon LLC. Machine learning on aws. <https://aws.amazon.com/machine-learning>, 2014. Accessed: 2018-05-04.
- [51] Tianyi Chen, Frank E. Curtis, and Daniel P. Robinson. A reduced-space algorithm for minimizing ℓ_1 -regularized convex functions. *SIAM Journal on Optimization*, 27(3):1583–1610, 2017.
- [52] Tianyi Chen, Frank E. Curtis, and Daniel P. Robinson. FaRSA for ℓ_1 -regularized convex optimization: local convergence and numerical experience. *Optimization Methods and Software*, 33(2):396–415, 2018.
- [53] Dimitri P Bertsekas. *Convex optimization theory*. Athena Scientific Belmont, 2009.
- [54] Dimitri P. Bertsekas, Angelia Nedi, and Asuman E. Ozdaglar. *Convex analysis and optimization*. 2003.
- [55] Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2012.
- [56] Patrick L. Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer, 2011.
- [57] Amir Beck and Marc Teboulle. Gradient-based algorithms with applications to signal recovery. *Convex optimization in signal processing and communications*, pages 42–88, 2009.

- [58] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.
- [59] Zdenek Dostál. Box constrained quadratic programming with proportioning and projections. *SIAM Journal on Optimization*, 7(3):871–887, 1997.
- [60] Zdenek Dostál. A proportioning based algorithm with rate of convergence for bound constrained quadratic programming. *Numerical Algorithms*, 34(2-4):293–302, 2003.
- [61] Zdenek Dostál and Joachim Schoberl. Minimizing quadratic functions subject to bound constraints with the rate of convergence and finite termination. *Computational Optimization and Applications*, 30(1):23–43, 2005.
- [62] Ana Friedlander, José Mario Martínez, and Marcos Raydon. A new method for large-scale box constrained convex quadratic minimization problems. *Optimization Methods and Software*, 5(1):57–74, 1995.
- [63] Ana Friedlander and José Mario Martínez. On the numerical solution of bound constrained optimization problems. *RAIRO-Operations Research*, 23(4):319–341, 1989.
- [64] Ana Friedlander and José Mario Martínez. On the maximization of a concave quadratic function with box constraints. *SIAM Journal on Optimization*, 4(1):177–192, 1994.
- [65] Yu-Hong Dai, William W. Hager, Klaus Schittkowski, and Hongchao Zhang. The cyclic barzilai—borwein method for unconstrained optimization. *IMA Journal of Numerical Analysis*, 26(3):604–627, 2006.

- [66] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.
- [67] Ron S. Dembo, Stanley C. Eisenstat, and Trond Steihaug. Inexact newton methods. *SIAM Journal on Numerical analysis*, 19(2):400–408, 1982.
- [68] Jon L. Bentley and Robert Sedgewick. Fast algorithms for sorting and searching strings. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 360–369. Society for Industrial and Applied Mathematics, 1997.
- [69] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley Professional, 2011.
- [70] Ake Bjorck. *Numerical methods for least squares problems*. Siam, 1996.
- [71] Gene H. Golub and Charles F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.
- [72] M. Morris Mano. *Digital design*. EBSCO Publishing, Inc., 2002.
- [73] Man-Chung Yue, Zirui Zhou, and Anthony Man-Cho So. A family of inexact sqa methods for non-smooth convex minimization with provable convergence guarantees. *arXiv preprint arXiv:1605.07522*, 2016.
- [74] Mark Schmidt. Graphical model structure learning with l1-regularization. *University of British Columbia*, 2010.
- [75] Mark Schmidt, Glenn Fung, and Romer Rosales. Optimization methods for l1-regularization. *University of British Columbia, Technical Report TR-2009*, 19, 2009.
- [76] Mark Schmidt, Glenn Fung, and Rmer Rosales. Fast optimization methods for l1 regularization: A comparative study and two new approaches.

In *European Conference on Machine Learning*, pages 286–297. Springer, 2007.

- [77] Shirish Krishnaji Shevade and S Sathya Keerthi. A simple and efficient algorithm for gene selection using sparse logistic regression. *Bioinformatics*, 19(17):2246–2253, 2003.
- [78] Kaggle. Avazu’s click-through prediction. <https://www.kaggle.com/c/avazu-ctr-prediction/data>, 2014. Accessed: 2018-05-04.
- [79] Libsvm. Winning solution. <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#avazu>, 2014. Accessed: 2018-05-04.
- [80] William H. Wolberg and Olvi L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the national academy of sciences*, 87(23):9193–9196, 1990.
- [81] Andrew V. Uzilov, Joshua M. Keegan, and David H. Mathews. Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. *BMC bioinformatics*, 7(1):173, 2006.
- [82] Uri Alon, Naama Barkai, Daniel A Notterman, Kurt Gish, Suzanne Ybarra, Daniel Mack, and Arnold J Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.
- [83] Mike West, Carrie Blanchette, Holly Dressman, Erich Huang, Seiichi Ishida, Rainer Spang, Harry Zuzan, John A. Olson, Jeffrey R. Marks, and Joseph R. Nevins. Predicting the clinical status of human breast

- cancer by using gene expression profiles. *Proceedings of the National Academy of Sciences*, 98(20):11462–11467, 2001.
- [84] Tin Kam Ho and Eugene M. Kleinberg. Building projectable classifiers of arbitrary complexity. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 2, pages 880–885. IEEE, 1996.
- [85] Vincent G. Sigillito, Simon P. Wing, Larrie V. Hutton, and Kile B. Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10(3):262–266, 1989.
- [86] Hsiang-Fu Yu, Hung-Yi Lo, Hsun-Ping Hsieh, Jing-Kai Lou, Todd G McKenzie, Jung-Wei Chou, Po-Han Chung, Chia-Hua Ho, Chun-Fu Chang, Yin-Hsuan Wei, et al. Feature engineering and classifier ensemble for kdd cup 2010. In *KDD Cup*, 2010.
- [87] Todd R. Golub, Donna K. Slonim, Pablo Tamayo, Christine Huard, Michelle Gaasenbeek, Jill P. Mesirov, Hilary Coller, Mignon L. Loh, James R. Downing, Mark A. Caligiuri, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science*, 286(5439):531–537, 1999.
- [88] S. Sathiya Keerthi and Dennis DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6(Mar):341–361, 2005.
- [89] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50. ACM, 2016.

- [90] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.
- [91] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 681–688. ACM, 2009.
- [92] John C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods*, pages 185–208, 1999.
- [93] David Martin Powers. Evaluation: from precision recall and f-measure to roc informedness markedness and correlation. 2011.
- [94] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.
- [95] Arnau Tibau Puig, Ami Wiesel, and Alfred O. Hero. A multidimensional shrinkage-thresholding operator. In *Statistical Signal Processing, 2009. SSP’09. IEEE/SP 15th Workshop on*, pages 113–116. IEEE, 2009.
- [96] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.
- [97] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

- [98] Pablo Sprechmann, Ignacio Ramirez, Guillermo Sapiro, and Yonina Eldar. Collaborative hierarchical sparse modeling. In *Information Sciences and Systems (CISS), 2010 44th Annual Conference on*, pages 1–6. IEEE, 2010.
- [99] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. Group lasso with overlap and graph lasso. In *Proceedings of the 26th annual international conference on machine learning*, pages 433–440. ACM, 2009.
- [100] Peng Zhao, Guilherme Rocha, and Bin Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, pages 3468–3497, 2009.
- [101] Sahand Negahban, Bin Yu, Martin J. Wainwright, and Pradeep K Ravikumar. A unified framework for high-dimensional analysis of m -estimators with decomposable regularizers. In *Advances in Neural Information Processing Systems*, pages 1348–1356, 2009.
- [102] Leonhard James Rogers. An extension of a certain theorem in inequalities. *Messenger Math*, 17:145–150, 1888.
- [103] Francis Bach, Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, et al. Convex optimization with sparsity-inducing norms. *Optimization for Machine Learning*, 5:19–53, 2011.
- [104] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.
- [105] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l_1 -ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.

- [106] Laurent Condat. Fast projection onto the simplex and the ℓ_1 ball. *Mathematical Programming*, 158(1-2):575–585, 2016.
- [107] Tianyi Chen and Xiang Hao. Personalized recommendation system framework. *Internal Paper*, 2016.
- [108] Wikipedia. Moore’s law. https://en.wikipedia.org/wiki/Moore%27s_law. Accessed: 2018-05-23.
- [109] Amazon LLC. Amazon web services (aws) - cloud computing services. <https://aws.amazon.com/>. Accessed: 2018-05-23.
- [110] Microsoft Azure. Microsoft azure cloud computing platform & services. <https://azure.microsoft.com/en-us/>. Accessed: 2018-05-23.
- [111] David B. Skillicorn. Models for practical parallel computation. *International Journal of Parallel Programming*, 20(2):133–158, 1991.
- [112] OpenMP Architecture Review Board. Openmp. <https://www.openmp.org/>. Accessed: 2018-05-23.
- [113] Intel. Cilk plus. <https://www.cilkplus.org/>. Accessed: 2018-05-23.
- [114] Blaise Barney. Openmp tutorial. <https://computing.llnl.gov/tutorials/openMP/>. Accessed: 2018-05-23.
- [115] Richard L Graham, Timothy S Woodall, and Jeffrey M Squyres. Open mpi: A flexible high performance mpi. In *International Conference on Parallel Processing and Applied Mathematics*, pages 228–239. Springer, 2005.
- [116] Intel. Intel mpi. <https://software.intel.com/en-us/intel-mpi-library>. Accessed: 2018-05-23.

- [117] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.
- [118] Chad Scherrer, Mahantesh Halappanavar, Ambuj Tewari, and David Haglin. Scaling up coordinate descent algorithms for large ℓ_1 -regularization problems. *arXiv preprint arXiv:1206.6409*, 2012.
- [119] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin. Feature clustering for accelerating parallel coordinate descent. In *Advances in Neural Information Processing Systems*, pages 28–36, 2012.
- [120] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for ℓ_1 -regularized loss minimization. *arXiv preprint arXiv:1105.5379*, 2011.
- [121] Zhimin Peng, Ming Yan, and Wotao Yin. Parallel and distributed sparse optimization. In *Signals, Systems and Computers, 2013 Asilomar Conference on*, pages 659–646. IEEE, 2013.
- [122] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I. Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014.
- [123] Virginia Smith, Simone Forte, Michael I. Jordan, and Martin Jaggi. ℓ_1 -regularized distributed optimization: A communication-efficient primal-dual framework. *arXiv preprint arXiv:1512.04011*, 2015.

- [124] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.
- [125] Yong Zhuang, Yuchin Juan, Guo-Xun Yuan, and Chih-Jen Lin. Naive parallelization of coordinate descent methods and an application on multi-core l1-regularized classification.
- [126] Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7(Jul):1467–1492, 2006.

Vita

Tianyi Chen was born in August, 1990 in China. He studied at Dalian University of Technology in China from 2009 to 2014, where he graduated with a Bachelor of Science in Applied Mathematics, and attended the Complex Network Laboratory to perform research on multi-agent systems. From 2014 to 2018, at Johns Hopkins University, he enrolled in the Ph.D. program at the Department of Applied Mathematics and Statistics, during which time he also obtained a Master of Science in Engineering from the Department of Computer Science. His graduate research at Johns Hopkins University mainly focused on designing, analyzing, and developing high-performance optimization algorithms. During 2016 and 2017, he attended the summer research programs at Amazon and Microsoft to build scalable algorithms based on various optimization methods. Tianyi is currently working as a Software Development Engineer, Research & Development at Microsoft Cloud & AI.